IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor:	Elias Levy	Examiner:	Christopher J. Brown
Application No.:	10/775,764	Art Unit:	2434
Filed:	February 9, 2004	Docket No.:	SYMAP042
Title:	CAPTURING A SECURITY BREACH		

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in a prepaid envelope addressed to: Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on:

3/23, 2009.

Elaine Nguyen

DECLARATION UNDER 37 CFR § 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir or Madam:

I, Elias Levy, declare as follows:

1. I am a Senior Technical Director in the Security Technology and Response group at Symantec Corporation. I am the named inventor in the above-referenced patent application.

2. The technology disclosed in the above-referenced patent application was embodied in a system called the DeepSight Attack Quarantine System (AQS), which was developed for use by Symantec to collect information about network security threats.

3. I have reviewed the above patent applications and the claims as amended in Amendment B thereto. The technology recited in the claims was embodied in the DeepSight Attack Quarantine System (AQS).

4. Attached hereto as Exhibit A is a copy of a Functional Requirements document entitled DeepSight Attack Quarantine System Version 1.0, Documentation Revision 1.21, dated November 18, 2002. The Functional Requirements document describes, for example at page 9, Section 3, performing the steps of deploying a honey pot, detecting a breach of the honey pot, capturing (saving) a state of the honey pot, and reinitializing and

redeploying the honey pot. The Functional Requirements document further describes, e.g., at pages 21-27 (Section 5.1.4), an implementation of the above. At pages 22-23, for example, the commands "dionaea-hp-start" (to start a honey pot instance) and "dionaea-hp-start-vmware" (to start a honey pot instance's virtual host) are described. At pages 26-27 the command "dionaea-monitord", for determining when a honey pot instance has been breached and when to suspend at restart it, is described. At page 27, the command "dionaea-hp-suspend" is described as being invoked when a suspend criterion associated with a breach is met. The "dionaea-hp-suspend" command is described at pages 23-24, and includes moving the honey pot instance's directory to a post-intrusion analysis database.

5. I have reviewed email and electronic calendar records and located the following references to the project described in Exhibit A:

a. An October 15, 2002 email sent by me proposing the Attack Quarantine System described in Exhibit A and internally named "dionaea".

b. A November 20, 2002 email from Mario van Velzen to Craig Davison and myself sending the latest version of the product requirements document.

c. A number of email messages sent by Craig Davison throughout December 2002 discussing development.

d. An email from me on December 30, 2002, sending out the latest functional specification draft.

e. A entry on my calendar for December 31, 2002 for a meeting regarding AQS development.

f. An email from January 8, 2003, that mentions setting up a Microsoft Project plan for the AQS development.

g. A discussion via emails sent on January 15, 2003, with Mario van Velzen and Craig Davison planning the development of the Attack Quarantine System described in Exhibit A.

h. A message from me on January 20, 2003 sending out version 1.0 of the functional specification.

i. A recurring calendar entry reflecting weekly meetings regarding AQS development beginning in February 2003.

6. I have reviewed the applicable source code repository records, and those records indicate that source code to implement the DeepSight AQS was first checked into the system on February 3, 2003, the date on which a new source code repository was created for the project, with numerous and regular updates checked in through June 27, 2003. Substantial work on source code to implement the Functional Requirements set out in Exhibit A would have been completed prior to the new source code repository being created on February 3, 2003.

7. Attached hereto as Exhibit B is a copy of a presentation entitled DeepSight Attack Quarantine System (AQS), prepared by myself for presentation at Symantec's 2003 Architects Conference. The associated electronic file, a Microsoft PowerPoint (.ppt) file, shows a date of October 8, 2003 in its file properties. The slides include a screenshot that indicates the DeepSight AQS was deployed by July 28, 2003 (see page entitled "Honey Pots : Work Queue Management) and the slide entitled "AQS v1 Statistics indicates the AQS "Entered production in July 2003". The system referred to in the presentation implemented the Functional Requirements referenced above and attached hereto as Exhibit A.

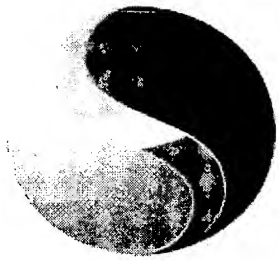
I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

~~Elias Levy~~

March 20, 2009

Date

Exhibit A



symantec™

Confidential

Functional Requirements

DeepSight Attack Quarantine System Version 1.0
Documentation Revision 1.21

Written by : Elias Levy
Contributors : Craig Davison, Mario Van Velzen

Creation Date : 2002/11/18

Table of Contents

1. Introduction

1.1 Document Purpose and Audience

The purpose of this document is to describe the architecture and specification of the DeepSight Attack Quarantine System (AQS) version 1.0. It will provide the development team with the details necessary to estimate and code the application. It will provide QA team the details necessary to write test plans and testing frameworks.

The audience for this document includes:

- Product management
- The development team
- The maintenance team
- The testing team
- The communications team responsible for help and training materials

1.2 Scope and Description of the Module

Within the scope of this document is the description of the functional requirements of the core subsystems of the Attack Quarantine System: VMware-based Honey Pot Server, Honey Pot Monitoring & Management subsystem Network Address Translation subsystem, Network Access Control subsystem, Network Traffic Capture subsystem, Honey Pot Support Services subsystem, Post-Intrusion Automated Analysis subsystem, and Web-based Interface.

How to install an operating system and/or applications to a VMware virtual disk, what operating system and/or applications to install, or how to configure them, is outside of the scope of this document, except for the requirements set for in Section 5.1.1.

The specification of the Malware Oracle system is outside of the scope of this document. Please, refer to the Malware Oracle Functional Specification Document.

The specification of the Inquisitor system is outside of the scope of this document. Please, refer to the Inquisitor Functional Specification Document.

The specification of the Digital Immune System is outside of the scope of this document. Please, refer to the document Immune System network protocol, twelfth draft.

1.3 Performance Criteria

The VMware honey pot server must be capable of executing eight virtual honey pot systems concurrently.

The web-based interface must be capable of serving twenty concurrent operators.

The post-intrusion automated analysis subsystem must be capable of analyzing ten breaches per hour.

The network address translation subsystem must be capable of handling 65,536 addresses,

The network capture subsystem must be capable to records network traffic without any packet loss.

1.4 Standards

The *pcap* dump file format is a defacto standard. While no standard documents the format it is documented by the *pcap* library sources.

The use of Microsoft SQL Server data types and stored procedures breaks standard compliance with the SQL standard.

1.5 International Implications

Special care must be taken when handling and displaying data from honey pot systems that use a different character set and character encoding (e.g. Windows NT/2000/XP systems and Unicode).

1.6 Additional Documentation

These are the additional documents that describe the module or other modules it interacts with.

Document Name	Description
DeepSight AQS 1.0 Business Requirements Document	The business requirements for the module.
DeepSight Malware Oracle 2.0 Functional Requirements Document	The functional requirements for the Malware Oracle system.
DeepSight Inquisitor 1.0 Functional Requirements Document	The functional requirements for the Inquisitor system.
Immune System network protocol, twelfth draft	DIS communications protocol specification.

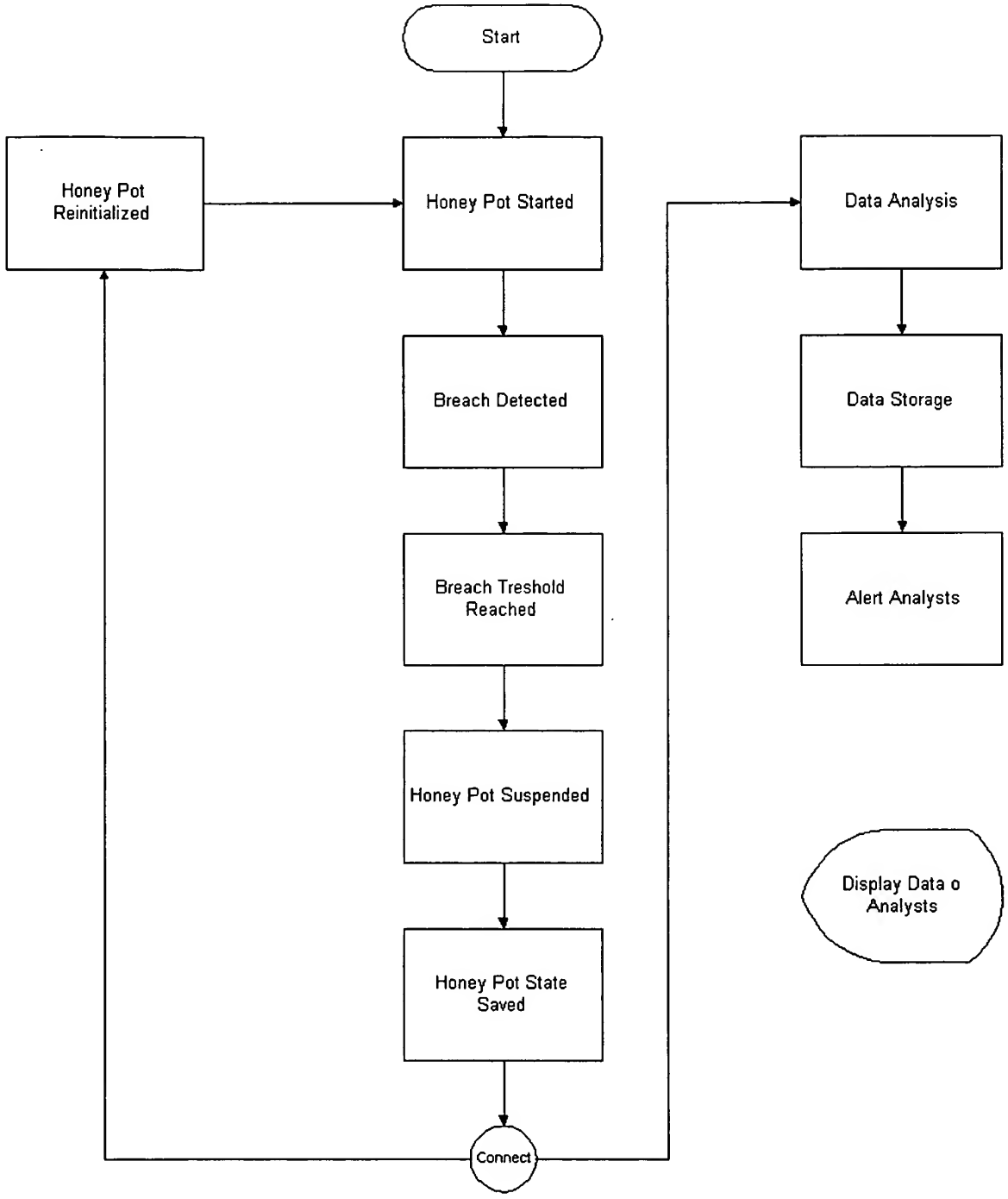
2. Definitions, Acronyms and Abbreviations

The following is a list of some of the acronyms and terms used in this module (and in the document) and their definitions.

Term	Definition
AQS	Attack Quarantine System
DIS	Digital Immune System
TMS	Threat Management System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
NAT	Network Address Translation
Honey Pot Archetype	The honey pot "template" from a honey pot system is derived. A honey pot archetype can be used to derive multiple honey system, which can execute multiple times in parallel.
Honey Pot System	A honey pot derived from an archetype. A honey pot system may be executed multiple times serially.
Honey Pot Instance	One execution of a honey pot system.

3. Work Flow

The following diagram describes the general workflow of a single honey pot system within the architecture:



4. Security

By its very nature the AQS has a number of highly exposed components. For this reason the AQS network must not be connected to any Symantec production, development, or testing network. The AQS network is solely connected to the Internet.

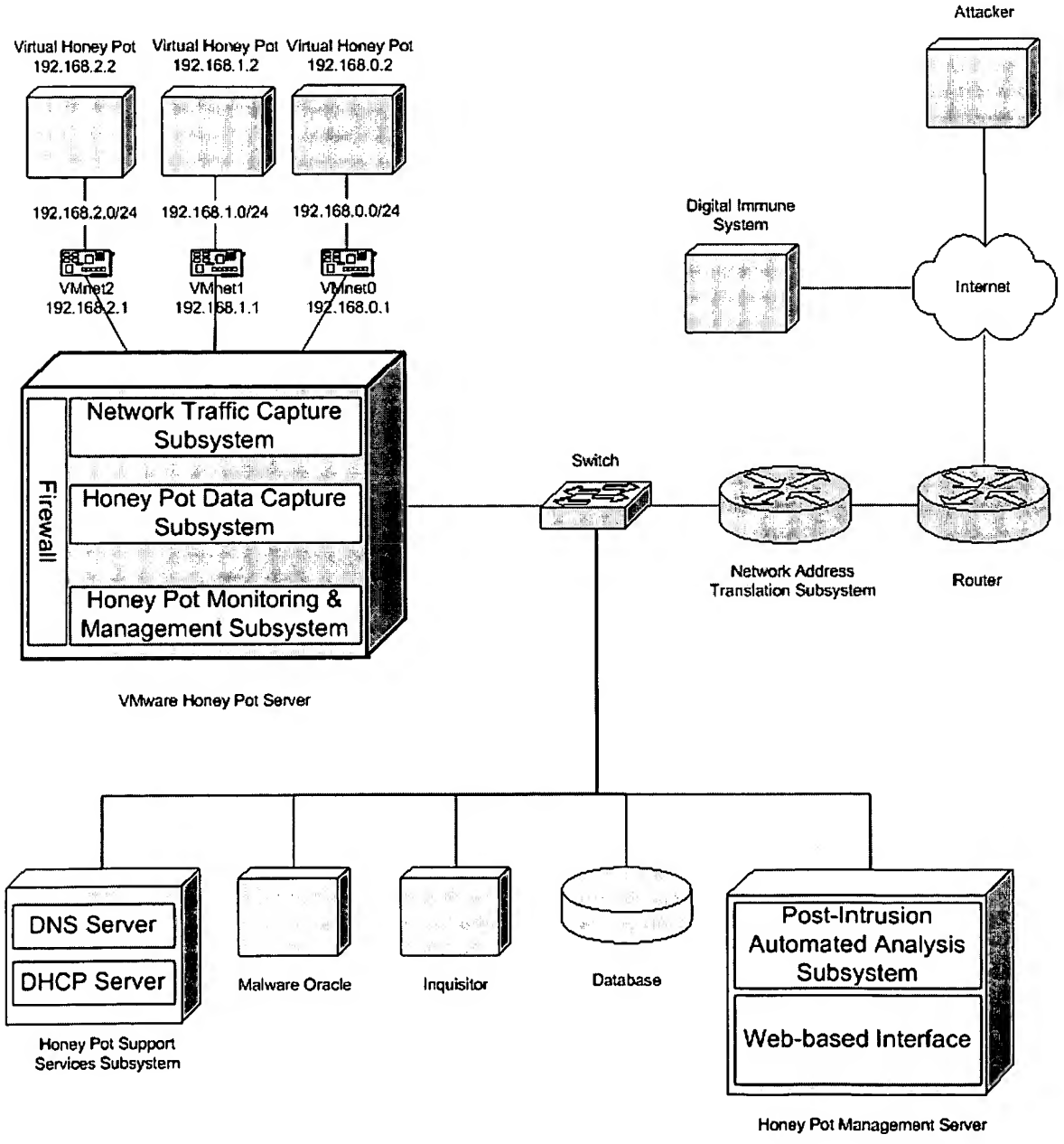
Within the AQS some components are meant to be breached, while others are meant to be secure. A subsystem, the Network Access Control subsystem, contains network traffic from the breached systems. Furthermore, the secure hosts are hardened against attack.

5. Architecture

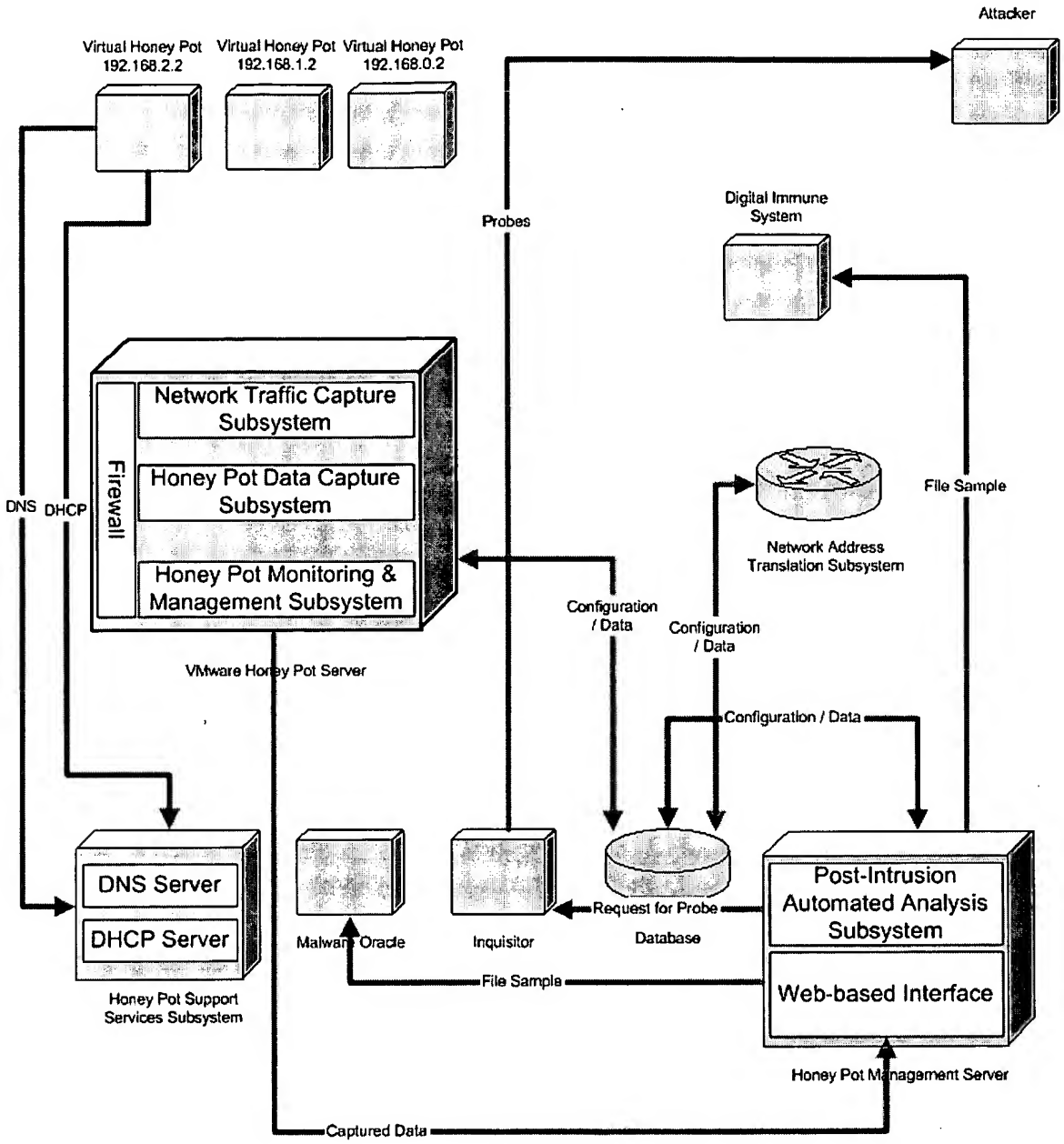
The Attack Quarantine System (AQS) version 1.0 product is comprised by a number of different components: a honey pot server that hosts the virtualization software, honey pot monitoring & management subsystem, honey pot network traffic capture subsystem, and network access control subsystem; a network address translation subsystem; a honey pot support subsystem; a database; and a honey pot management server that hosts the post-intrusion analysis subsystem and the web-based interface.

The AQSv1 interacts with a number of external systems: the Internet, the DeepSight Malware Oracle, DeepSight Inquisitor, and the Digital Immune System (DIS).

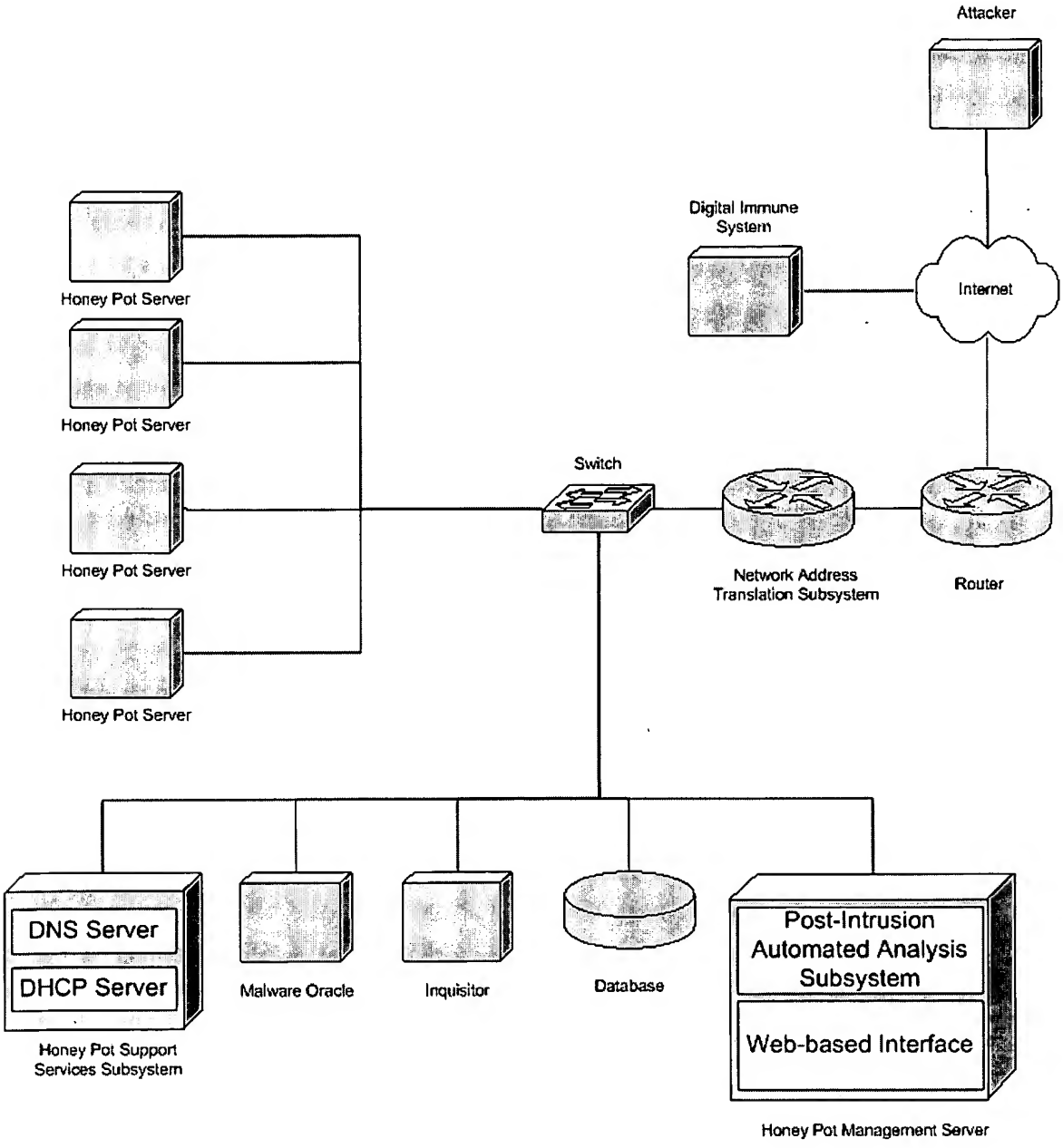
The following diagram shows the AQS components and network topology.



The following diagram shows the AQS components and their interactions.



Multiple honey pot servers can be managed by a single honey pot management server, store data in a single database, and feed samples to DIS. The following diagram illustrates this configuration.



5.1 VMware Honey Pot Server

The VMware honey pot server is the hearth of AQSv1. The VMware honey pot server executes a number of virtual honey pots and hosts a number of different subsystems.

The subsystems the VMware honey pot server hosts are:

- Virtualization Software, which executes the virtual honey pot system instances.

- Network Access Control (firewall), to isolate honey pot systems from each other,
- Network Traffic Capture, which records network traffic to and from the honey pot systems.
- Honey Pot Monitoring & Management, which determines when a honey pot system instance has been breached, saves its state, and initiates a new honey pot system instance.

All of the files that comprise the AQSv1 system in the VMware honey pot server are stored under the **/var/dionaea** directory. Executable components of AQSv1 the VMware honey pot server are found in the **/var/dionaea/bin** directory.

5.1.1 Virtualization Software

A honey pot archetype is a template that determines the type and configuration of a honey pot system. Multiple honey pot systems can be defined from the same honey pot archetype. A honey pot instance is an execution or instantiation of a honey pot system from its archetype. Honey pot instances from the same honey pot system must be executed sequentially. Honey pot instances from different honey pot systems can be executed in parallel.

The virtualization software creates virtual honey pot instances of honey pot systems based on their archetype. The virtualization software used in the VMware honey pot server is VMware GSX Server version 2.0 or later, running under the Linux operating system.

All the files that comprise a honey pot archetype are stored in the directory **/var/dionaea/archetypes/*<at>***, where ***<at>*** is the archetype id. All the files that comprise a honey pot system instance while it is executing are stored in the directory **/var/dionaea/honeypots/*<id>***, where ***<id>*** is the honey pot system instance id.

The file system data of a virtual honey pot system is stored in a single VMware virtual disk. A VMware virtual disk is composed of one or more virtual disk file, which have an extension of **vmdk**. The archetype VMware virtual disk files of a system are stored in the directory **/var/dionaea/archetypes/*<at>*/vmware** and copied to the honey pot instance's **/var/dionaea/honeypots/*<id>*** directory when the honey pot system instance is initialized.

The creation of these archetype virtual disks is outside of the scope of this document, except for the following requirements on the configuration of the software installed on them:

- Operating systems must be configured to use DHCP to discover and configure their network parameters, including their IP address, default gateway, and DNS servers.
- Other than for DHCP the operating systems and applications must be configured so that they do not initiate network activity when left in an idle state. If the system still produces some network traffic it must be described in the form of a *pcap* filter in a file with the extension **nign**. This file is stored in the archetype **/var/dionaea/archetypes/*<at>*** directory.

- Any web servers making use of HTTP over SSL/TLS (HTTPS) must be configured to use the same private key, so that their traffic can be decrypted an analyzed by network monitoring tools.

The virtual machine configuration data for a honey pot achetype is stored in a single VMware configuration file, which has an extension of **cfg**. This file is store in the archetype directory **/var/dionaea/archetypes/<*at*>/vmware**.

Each virtual honey pot system must be configured to have at least:

- An 8-bit video adapter.
- A PS/2 keyboard.
- A PS/2 mouse.
- A single undoable IDE virtual disk of 4GB.
- A single unconnected floppy drive.
- A single unconnected CD-ROM drive.
- A single network interface connected to a host-only network shared with no other virtual honey pot.

It should also be configured to use the minimum amount of memory necessary to operate correctly.

5.1.2 Network Access Control

The Network Access Control (firewall) subsystem denies some network traffic from/to honey pot systems.

The firewall:

- Drops any spoofed packet from the honey pot systems.
- Drops any cross-honey pot traffic.
- Denies any packets to any of the honey pot server addresses
- Denies traffic to any addresses in the support services network, except for traffic destined to the Honey Pot Support services.
- Log all connection initiations to and from the honey pot systems and the honey pot server.

5.1.2.1 Reasoning

Denying Spoofed Traffic

It is common for breached hosts to be used for denial of service (DoS) attacks. Often these DoS attacks make use of spoofed packets. By dropping any spoofed packets that originate from the honey pot systems we will mitigate the chance that they will be used to attack a third-party.

Denying Cross-Honey Pot Traffic

Malicious code or an attacker that has breached a honey pot instance may attempt to attack other honey pot instances since they are near each other in the IPv4 address space. The new honey pot instance it may choose to attack may already be under attack by some other attacker. The new attack against the other honey pot instance would contaminate the data associated with the previous attack against it. By dropping any cross-honey pot traffic we insure that an attack against one honey pot instance will not contaminate data about a concurrent attack against a different honey pot instance.

Denying Traffic From Different Remote Host To One Honey Pot Instance

This leaves open the possibility of two distinct external attackers launching attacks against the same honey pot instance concurrently, contaminating each other's data. By monitoring for the first incoming connection to the honey pot instance and dropping any packets that do not originate from the same source IPv4 address we can eliminate this contamination, at the expense of capturing attacks that involve more than one source address. Since the likelihood of two simultaneous attacks is low we have decided not to impose this filtering.

Denying Traffic To The Honey Pot Server

The honey pot server's security is paramount for the proper functioning of the AQS. An attacker that is trying to breach, or has breached, a honey pot instance, is likely to attempt to breach the honey pot server. To minimize the likelihood of such an occurrence we deny all network traffic to the honey pot server.

Denying Traffic To The Honey Pot Support Network

The systems in the honey pot support network (e.g. database, honey pot management server) are important components in the proper functioning of the AQS. An attacker that is trying to breach, or has breached, a honey pot instance, is likely to attempt to breach the system in the honey pot support network. To minimize the likelihood of such an occurrence we deny all network traffic to from the honey pot systems to the honey pot support network, except for traffic destined for the Honey Pot Support Services subsystem.

Connection Logging

While a number of other subsystems are responsible for capturing network traffic and monitoring the initiation of new connections to and from honey pots systems they all depend on some daemon to be executing. By using the kernel's ability to log connection initiations to and from honey pots systems we have a back up in the event of failure by the other subsystem's daemons.

5.1.2.2 Implementation Details

The Linux kernel, via the *netfilter* project, includes packet-filtering capabilities. Thus, the low-level details of network access control are already taken care of.

Denying Spoofed Traffic

We can drop spoofed packets originating from the honey pot systems by adding a rule to the *filter* table's *FORWARD* chain with a target of *DROP* for any packets coming in via any network interface, not including the one connected to the Internet, that have an IPv4 source address that is not within the IPv4 address space assigned to the honey pot server.

Denying Cross-Honey Pot Traffic

We can drop all cross-honey pot traffic by adding a rule to the *filter* table's *FORWARD* chain with a target of *DROP* for any packets with both an IPv4 source address and a destination address within the IPv4 address space assigned to the honey pots. This will not deny legitimate traffic from the honey pot's to the locally hosted DHCP relay agent, and vice versa, since the fate of such packets is determined by the *filter* table's *INPUT* and *OUTPUT* chains, and not by the *FORWARD* chains.

Denying Traffic To The Honey Pot Server

To deny traffic to the honey pot server, except that described bellow, we insert as the last rule in the *filter* table's *INPUT* chain a rule that matches all traffic and has a target of *DENY*.

Permitting Traffic To The DHCP Support Service

To relay DHCP traffic to the DHCP support service the honey pot server executed a DHCP relay agent. This service listens to UDP port 67.

To allow DHCP requests from honey pot systems to the DHCP relay agent we insert a rule in the *filter* table's *INPUT* chain that matches the destination UDP port of *67*, any interface, not including the one connected to the Internet, and a connection state of *NEW,ESTABLISHED* with a target of *ACCEPT*.

To allow DHCP requests from the DHCP relay agent to the DHCP server we insert a rule in the *filter* table's *INPUT* chain that matches the destination UDP port of *67*, the Internet connected interface, source IPv4 address that matches the IPv4 address of the system hosting the Honey Pot Support Services subsystem, and a connection state of *ESTABLISHED* with a target of *ACCEPT*.

Denying Traffic To The Honey Pot Support Network

To deny traffic to the honey pot support network, except that described bellow, we insert as the last rule in the *filter* table's *FORWARD* chain a rule that matches a destination IPv4 address within the honey pot support network address range, and a target of *DENY*.

Permitting Traffic To The DNS Support Service

To allow DNS queries from the honey pot systems to the caching DNS server we insert a rule in the *filter* table's *FORWARD* chain that matches the UDP destination port number of 53, any interface, not including the one connected to the Internet, a destination IPv4 address that matches the IPv4 address of the system hosting the Honey Pot Support Services subsystem and a connection state of *NEW* with a target of *ACCEPT*.

To allow DNS responses from DNS servers in the Internet to the caching DNS server we insert a rule in the *filter* table's *INPUT* chain that matches the UDP source port number of 53, the UDP destination port number of 53, the interface connected to the Internet, and a connection state of *ESTABLISHED* with a target of *ACCEPT*.

Record of New Connections

We can log all connection initiations to and from the honey pot systems, and to and from the honey pot server, by adding rules to the *mangle* table's *PREROUTING* chain with a target of *LOG* that match the connection state of *NEW* for all interfaces in the system.

5.1.3 Network Traffic Capture

The Network Traffic Capture subsystem records all network traffic from/to a honey pot.

The subsystem:

- Records all network traffic from/to a honey pot instance.

5.1.3.1 Reasoning

Homegrown Network Traffic Capture Software

There are a number of tools that allow us to capture network traffic (e.g. tcpdump or ethereal). Most of these tools are based in the *pcap* packet capture library and provide a rich language to determine what traffic to capture. Yet, all of them provide nothing more than a rudimentary method to store captured network traffic: a single capture file for all captured traffic.

We wish to capture traffic to multiple honey pot instances. For convenience's sake we want captured network traffic to/from different honey pot instances to be stored in different files. We also want to be able to tell the network traffic capture software when to start and top capturing traffic for any particular honey pot instance. We do not wish to capture network traffic towards a honey pot system while an instance of it does not exist.

Given our requirements and the limitations of the available tools we need to develop our own network traffic capture software that can be directed to start and stop capturing network traffic to and from specific IP addresses, and can store the captured network traffic in different files based on the matching IP address.

Capture File Format

The *pcap* dump file format is understood by a large number of third-party tools that we can use to analyze the network traffic. For this reason we store captured network traffic in file that use the *pcap* dump file format.

Where to Capture Traffic

Under Linux the network capture hook captures an outgoing packet right before it is placed on the wire, and captures an incoming packet right before it is handed to the network protocol stack. For packets that are being forwarded this means we have a choice of where to capture the packet.

Because we are performing some packet filtering, a packet received in one interface for forwarding may not make it out the other interface. Additionally, packets may be directed to the honey pot server itself. Such packets can only be captured via the interface they arrived.

Since we want to capture as much data as possible about the network traffic sent by a honey pot system or to a honey pot system we capture network traffic as it is received or sent by the network interface connected to the honey pot system's network.

Determining What Traffic To Capture

We could use a honey pot system's private IPv4 address to capture traffic towards and from it, at the interface that connects to the honey pot network, but such an approach would fail to capture some network traffic.

If malicious code in the honey pot instance sends spoofed IPv4 traffic the filter would fail to match it and capture it. Such filter would also fail to capture any broadcast IPv4 traffic and any non-IPv4 traffic, such as ARP or NetBEUI traffic.

To solve this problem we could instead capture traffic that matches honey pot system's Ethernet address in the source or destination address of an Ethernet frame, but this option suffer from the problem that Ethernet frames are easily spoofed as well.

The solution is to capture all network traffic sent or received via the network interface associated with the honey pot system.

PF_PACKET vs. pcap

If we were to use the *pcap* library for capturing packets this would require the use of multiple *pcap packet capture descriptors* (*pcap_t*), since a *pcap* packet capture descriptor can only be associated with a single interface, but the *pcap* library has no function that can poll multiple packet capture descriptors at the same time.

We would thus need to continuously loop reading from a non-blocking packet capture descriptor draining processor resources, or we would need to create one thread per packet capture descriptor. If we go with a threaded approach then we must determine how to handle the SIGHUP signal that we use to command the daemon to rescan the configuration directory, and how to interrupt the threads that are reading from the packet capture descriptor.

Instead, we bypass the *pcap* library and create *PF_PACKET* sockets using an *ETH_P_ALL* protocol bound to the network interface associated with the honey pot system. This permits us to capture all traffic associated with the honey pot coming in via this interface regardless of Ethernet or IPv4 addresses, while at the same time we us the *poll(2)* system call to wait on multiple descriptors simultaneously with a single thread.

5.1.3.2 Implementation Details

A command, **dionaea-start-netcap**, commands the **dionaea-netcapd** daemon to start capturing network traffic to or from a honey pot system. It takes as an argument the honey pot system's id. The command:

- Creates a zero length file in the directory **/var/dionaea/netcap/control** with a filename equal to the honey pot system's id.
- Reading the **dionaea-netcapd** daemon process id number from the file **/var/dionaea/netcap/pid**.
- Sending a SIGHUP signal to the daemon.

A daemon, **dionaea-netcapd**:

- Writes its process id to the file **/var/dionaea/netcap/pid**.
- Reads the directory **/var/dionaea/netcap/control**.
- For each honey pot system id read from the directory **/var/dionaea/netcap/control**, it:
    - Looks up its associated network interface in the **HoneyPotSystems** table.
    - Opens for writing a network capture file in the *pcap* dump file format named **/var/dionaea/netcap/partial/*<id>***, where ***<id>*** is the honey pot system's id, using the *pcap* library's *pcap_dump_open()* function.
    - Creates a *PF_PACKET* socket with a protocol of *ETH_P_ALL*.
    - Binds the socket to the network interface associated with the honey pot system.
    - Sets the socket to promiscuous mode.
- Registers a SIGHUP signal handler that:
    - Reads the directory **/var/dionaea/netcap/control**.
    - Compares the list of honey pot system ids this time the directory is read to the list of honey pot system ids the last time the directory was read.
    - For each honey pot system id that are no longer listed in the directory **/var/dionaea/netcap/control**, it:
        - Closes the *PF_PACKET* socket bound to the network interface associated with the honey pot system's id.
        - Closes the network traffic capture file **/var/dionaea/netcap/partial/*<id>***, using the *pcap* library *pcap_dump_close()*.
        - Looks up the honey pot's instance number in the **HoneyPotInstances** table.
        - Moves the file **/var/dionaea/netcap/partial/*<id>*** to **/var/dionaea/netcap/cap/*<id>*-*<instance>***, where ***<instance>*** is the honey pot instance number.
    - For each new honey pot system id listed in the directory **/var/dionaea/netcap/control**, it:
        - Looks up its associated network interface in the **HoneyPotSystems** table.
        - Opens for writing a network capture file in the *pcap* dump file format named **/var/dionaea/netcap/partial/*<id>***, where ***<id>*** is the honey pot system's id, using the *pcap* library's *pcap_dump_open()* function.
        - Creates a *PF_PACKET* socket with a protocol of *ETH_P_ALL*.

    - Binds the socket to the network interface associated with the honey pot system.
    - Sets the socket to promiscuous mode.
- Enters into a loop that:
    - Polls all *PF_PACKET* socket descriptors.
    - Captures network traffic from any descriptors with data.
    - Stores a captured packet in the appropiate *pcap* dump file using the *pcap* library's *pcap_dump()* function.

A command, **dionaea-stop-netcap**, commands the **dionaea-netcapd** daemon to stop capturing network traffic to or from some honey pot instance. The command:

- Deletes the zero length file in the directory **/var/dionaea/netcap/control** with a filename equal to the honey pot system's id.
- Reads the **dionaea-netcapd** daemon process id number from the file **/var/dionaea/netcap/pid**.
- Sends a SIGHUP signal to the daemon.

5.1.4 Honey Pot Monitoring & Management

The Honey Pot Monitoring & Management subsystem activates and deactivates honey pots, and determines when a honey pot has been breached.

The subsystem:

- Reads the desired honey pot status from the database.
- Stores the current honey pot status in the database.
- Instantiates a honey pot system.
- Instructs the Network Traffic Capture subsystem to start capturing network traffic to/from the honey pot instance.
- Determines if a honey pot instance has been breached.
- Determines whether a breached honey pot instance has reached a suspend condition.
- Suspends the execution of a honey pot instance.
- Instructs the Network Traffic Capture subsystem to stop capturing network traffic to/from the honey pot instance.
- Passes the collected data to the Post-Intrusion Automated Analysis subsystem.

5.1.4.1 Reasoning

The life cycle of a honey pot system – instantiation, breach, suspension – must be automated. There is also a need for the analysts to manually bring up or down a honey

pot instance, and for them to query the status of the honey pots system, from the analyst web-based interface. The analyst web-based interface must be capable of managing multiple honey pot servers.

Database

By using the database a the communication medium between the Honey Pot Monitoring & Management subsystem, the analyst web-based interface, and other subsystems we simplify the interdependencies of the system and at the same time provide long-term storage of the system's configuration.

Detecting a Breach

Version 1.0 of ASQ does not include instrumentation and data capture at the honey pot itself. The only data available to use to determine whether a honey pot instance has been breached is the network traffic to/from it. If the honey pot system's operating system and applications are configured so that they do not initiate network connections then any outgoing network connection from a honey pot instance is a good indication that the honey pot has been breached.

When to Suspend a Honey Pot Instance

We do not wish to suspend a honey pot instance that has been breached immediately. We want to provide sufficient time for the malicious code or attacker that breached the honey pot instance to reveal more about itself/himself. For example, an attacker may download a rootkit from some external server. After we have provided the attacker sufficient time to reveal more about himself, then we suspend the honey pot.

There are a number of ways to determine when to suspend the honey pot instance. We can provide the attacker with some maximum amount of time after which the honey pot instance is suspended. We can also track the number of outgoing network connections and suspend the honey pot instance after some number of them is reached. We combine these two methods so that the honey pot instance is suspended when either threshold is reached, and refine it by providing the attacker with a minimum amount of time.

5.1.4.2 Implementation Details

Dionaea-hp-start

The command, **dionaea-hp-start**, starts a honey pot instance. It takes as an argument the numeric id that identifies the honey pot system to instantiate. The command:

- Inserts a record in the **HoneyPotInstances** table with a status of *Starting*.
- Looks up the automatically assigned honey pot instance ID.
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Starting*.
- Creates the directory ***<id>*** in the directory **/var/dionaea/honeypots**, where ***<id>*** is the numeric id that identifies the honey pot system in the database.
- Executes **dionaea-start-netcap**.
- Executes **dionaea-hp-start-vmware**.

- Sleeps for 5 minutes.
- Updates the honey pot instance record in the HoneyPotInstances table with a status of *Executing*.
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Executing*.

Dionaea-hp-start-vmware

The command, **dionaea-hp-start-vmware**, starts a honey pot instance's virtual host. It takes as an argument the numeric id that identifies the honey pot system whose virtual host to start. The command:

- Creates the directory **vmware** in the directory **/var/dionaea/honeypots/<*id*>**.
- Determines the filename of the VMware archetype by looking it up in the **VMwareHoneyPotArchetypes** table.
- Copies the VMware configuration file **(cfg)** from the directory **/var/dionaea/archtypes/<*id*>/vmware** to the directory **/var/dionaea/honeypots/<*id*>/vmware**.
- Creates a symbolic link from the VMware virtual disk archetype files **(vmdk)** in the directory **/var/dionaea/archtypes/<*id*>/vmware** to the directory **/var/dionaea/honeypots/<*id*>/vmware**.
- Registers the honey pot's virtual machine with the VMware server by using the Perl *VMware::VmPerl::Server::register_vm()* subroutine.
- Determines the network interface connected to the honey pot by looking it up in the **HoneyPotSystems** table.
- Configures the honey pot virtual machine to use the network interface assigned to the honey pot system, by using the Perl *VMware::VmPerl::VM:set_config()* subroutine. Setting *ethernet0.present* to *TRUE*, *ethernet0.connectionType* to *custom*, and *ethernet0.vnet* to the interface name (e.g. */dev/vmnet0*).
- Determines the honey pot system's Ethernet address by looking it up in the **VMwareHoneyPotSystems** table.
- Assigns the honey pot virtual machine a new Ethernet address, by using the Perl *VMware::VmPerl::VM::set_config()* subroutine. Setting *ethernet0.address* to an Ethernet address of the form *00:50:56:XX:YY:ZZ*, where *XX* is a hex number between 00h and 3Fh, and *YY* and *ZZ* are hex numbers between 00h and FFh.
- Starting the honey pot's virtual machine, by using the Perl *VMware::VmPerl::VM::start()* subroutine.

Dionaea-hp-suspend

The command, **dionaea-hp-suspend**, suspends a honey pot instance. It takes as an argument the numeric id that identifies the honey pot system to suspend. The command:

- Modifies the record of the honey pot instance in the **HoneyPotInstances** table, setting the status to *Stopping*.
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Stopping*.
- Executes **dionaea-hp-suspend-vmware**.
- Executes **dionaea-stop-netcap**
- Moves the honey pot instance's directory from **/var/dionaea/honeypots/*<id>*** to **/var/dionaea/post/*<id>*-*<instance>***, where ***<instance>*** is the honey pot instance number as stored in the database.
- Moves and renames the file **/var/dionaea/netcap/cap/*<id>*-*<instance>*** to the file **/var/dionaea/post/*<id>*-*<instance>*/netcap**.
- Creates a tar archive of the **/var/dionaea/post/*<id>*-*<instance>*** directory named ***<id>*-*<instance>*.tar**.
- Copies the ***<id>*-*<instance>*.tar** to the host hosting the Post-Intrusion Automated Analysis subsystem under the directory **/var/dionaea/post/incoming** by executing the command **scp**.
- Removed the directory **/var/dionaea/netcap/post/*<id>*-*<instance>*** and the file ***<id>*-*<instance>*.tar**.
- Modifies the record of the honey pot instance in the **HoneyPotInstances** table, setting the instance number to the latest one and the status to *Suspended.*
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Suspended.*

Dionaea-hp-suspend-vmware

The command, **dionaea-hp-suspend-vmware**, suspends a honey pot instance's virtual host. It takes as an argument the numeric id that identifies the honey pot system whose virtual host to suspend. The command:

- Suspends the honey pot's virtual machine via the Perl *VMware::VmPerl::VM::suspend()* subroutine.
- Unregisters the honey pot's virtual machine with the VMware server by using the Perl *VMware::VmPerl::Server::unregister_vm()* subroutine.

Dionaea-hp-halt

The command, **dionaea-hp-halt**, halts a honey pot instance and throws away any captured data. It takes as an argument the numeric id that identifies the honey pot system to halt. The command:

- Modifies the record of the honey pot instance in the **HoneyPotInstances** table, setting the instance number to the latest one and the status to *Stopping*.
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Stopping*.
- Executes **dionaea-hp-halt-vmware**.
- Executes **dionaea-stop-netcap**
- Removes the honey pot instance's directory from **/var/dionaea/honeypots/*<id>***, and all its contents.
- Removes the directory **/var/dionaea/netcap/cap/*<id>*-*<instance>***, and all its contents.
- Modifies the record of the honey pot instance in the **HoneyPotInstances** table, setting the instance number to the latest one and the status to *Halted*.
- Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Halted*.

Dionaea-hp-halt-vmware

The command, **dionaea-hp-halt-vmware**, halts a honey pot instance's virtual host. It takes as an argument the numeric id that identifies the honey pot system whose virtual host to halt. The command:

- Suspends the honey pot's virtual machine via the Perl *VMware::VmPerl::VM::stop()* subroutine.
- Unregisters the honey pot's virtual machine with the VMware server by using the Perl *VMware::VmPerl::Server::unregister_vm()* subroutine.

Dionaea-managed

A daemon, **dionaea-managed**, determines when to start or halt a honey pot instance. The daemon:

- It loops forever, and:
    - Looks up the hostname of the system it runs in.
    - Determines whether there is any honey pot server for the system in the **HoneyPotServers** table, of type *VMware*, and its state.
    - If the state of the honey pot server is *Disabled* or *Archived*, then:

        - Looks up all honey pot system instances associated with the honey pot server with a status of *Starting*, *Executing*, *Waiting*, *Contacted* or *Breached*.
        - For each such honey pot server instance it executes the command **dionaea-hp-halt**.
    - Else, if the state of the honey pot server is *Enabled*, then:
        - Looks up all honey pot systems associated with the honey pot server, their state, and the state of their latest instance in the **HoneyPotSystems** and **HoneyPotIntances** table.
        - For each such honey pot server:
            - If the honey pot system state is *Enabled* and the status of the last honey pot instance is one of *Suspended*, *Halted*, *Analyzed* or *Reviewed*, then it executes the command **dionaea-hp-start** to start a new honey pot system instance.
            - If the honey pot system state is *Disabled* or *Archived* and the status of the last honey pot instance is one of *Executing*, Waiting, *Contacted* or *Breached*, then it executes the command **dionaea-hp-halt**.
    - Sleeps for 10 seconds.

Dionaea-monitord

A daemon, **dionaea-monitord**, determines when a honey pot instance has been breached and when to suspend it and start the post-intrusion analysis. The daemon:

- Inserts a rule in the *mangle* table's *FORWARD* chain that matches the interface facing the Internet as the outgoing interface and a state of NEW with a target of *QUEUE*.
- Loops and:
    - Waits for the kernel to pass it a packet that matches the inserted rule, signaling the initiation of a connection from a honey pot instance, by using the *libipq* C library or the *perlipq* Perl module, with a timeout of 5 seconds.
    - Lookup via what interface the packet arrived from.
    - Determines what honey pot system the interface is associated with by looking it up in the **HoneyPotSystems** table.
    - Increases the count of connections seen from the honey pot system's instance.
    - Responds to the kernel with a verdict of *NF_ACCEPT* for the packet, which will make the packet continue its route through the kernel connection tracking and packet filtering code.

- If this is the first connection:
  - Records the time the connection was seen.
  - Modifies the record of the honey pot instance in the **HoneyPotInstances** table, setting the instance number to the latest one and the status to *Breached.*
  - Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance state is now *Breached.*
- If, more than 5 minutes have elapsed since the first connection seen from the honey pot instance and, at least 10 network connections have been seen from the honey pot instance or at least 10 minutes have elapsed since the first connection seen from the honey pot instance, then:
  - Suspends the honey pot instance by executing **dionaea-hp-suspend.**
  - Zeros the connection counter for the honey pot system.
  -

5.1.5 Honey Pot Support Services

While the honey pot server does not itself host the Honey Pot Support Services subsystem its assistance is required to make the DHCP support service accessible to honey pot systems.

DHCP packets cannot be simply forwarded at the IP layer since some of them are broadcast. Instead the honey pot server must execute a DHCP relay agent. We make use of the ISC DHCP relay agent which listens to DHCP packets to UDP port 67 and relay them to the DHCP server in the system hosting the Honey Pot Support Services subsystem.

5.2 NAT Server

The Network Address Translation (NAT) server hosts the NAT subsystem. The subsystem maps one or more public IP addresses to each honey pot system's private IP address.

5.2.1 Network Address Translation

The Network Address Translation (NAT) subsystem maps a number of routable IPv4 addresses to the honey pot systems. It does this by mapping M-number of routable IPv4 addresses to each honey pot system, an M-to-N mapping.

Each honey pot system is assigned a percentage, or weight, that represents the portion of the available routable IPv4 addresses that must be mapped to it. The subsystem dynamically assigns available routable IPv4 addresses to each honey pot system in correspondence to the percentage, or weight, assigned to each.

When an incoming connection to a honey pot system is detected all IPv4 addresses mapped to the honey pot are unmapped from it and reassigned to other honey pot systems waiting for incoming connections.

The network address translation subsystem must also support as many high-level protocols as possible.

5.2.1.1 Reasoning

The virtualization host hosts the Network Address Translation (NAT) subsystem. Network-based worms and opportunistic attackers commonly select a networked system to attack by either randomly generating a target IPv4 address or randomly selecting a network and scanning a target network. In both cases, the chances of any one system being selected as the target for an attack increases with the number of unique routable IPv4 addresses the system posses.

Since the purpose of the AQS is the capture malicious code samples and attacks any factor that increases the likelihood of an AQS honey pot instance being targeted should be explored. In this case, it is very desirable for the AQS to have as large a network footprint as practically possible.

Seeing as each routable IPv4 address must be mapped to some honey pot instance we could choose to map them one-to-one. That is, one routable IPv4 address to one honey pot instance and vice versa. The disadvantage of this choice is that honey pot instances are resource intensive, and we only have resources to create, execute, and maintain a limited number of them.

A different choice is to map multiple routable IPv4 addresses to each honey pot instance, or an M-to-N map. This choice allows us to map as large an IPv4 address space as we wish to the limited number of honey pot instances we can support. This is the approach we use in AQS.

One troublesome aspect of the M-to-N approach is the question of what IPv4 addresses to use as the source IPv4 address for packets originating in the honey pot instance from the many assigned to it.

The issue can be easily resolves for packets that are part of a network "connection" that is initiated towards the honey pot instance. Since the connection is initiated towards the honey pot instance, the NAT system will use the destination IPv4 address of the packets that initiated the connection as the source IPv4 address for any packets that are part of the connection in the opposite direction. This is easily accomplished by maintaining state for TCP sessions and can be emulated to a reasonable degree for UDP and ICMP packets. Most NAT systems already support this "connection tracking" capability, including the Linux kernel, which we are making use of.

Packets that are part of a network connection that is initiated by the honey pot instance pose a bigger challenge. The destination IPv4 address is the only information the packet itself provides to help us decide which source IPv4 address to use. Since the destination IPv4 address may be different from the source IPv4 address of any connections initiated towards the honey pot instance is proves unreliable as a method to decide on a source IPv4 address.

One solution to this problem is to limit the choice of source IPv4 address by unmapping all but one IPv4 address from the honey pot instance after a connection has been initiated towards it, leaving mapped only the destination IPv4 address of this connection. Since honey pot systems must be configured so that they do not initiate network activity when left in an idle state any outgoing network connections can only be the result of some incoming network connection, and by that time the source IPv4 address of outgoing packets has already been selected.

This is the solution we use in AQS. When a honey pot system is instantiated one or more routable IPv4 addresses are mapped to the honey pot instance. When a connection is initiated towards the honey pot instance all routable IPv4 address except the one that is the destination IPv4 address of the connection are unmapped. When the honey pot is reinstantiated all routable IPv4 addresses are mapped once again.

One disadvantage of this approach is that between the time a connection towards the honey pot is initiated and the honey pot is reinstantiated any connection attempts to the unmapped IPv4 addresses are simply ignored, thus reducing the network footprint of AQS and its effectiveness.

To alleviate this we can increase the number of honey pot system, thus reducing the number of IPv4 addresses mapped to a single honey pot instance and contention for them. Another solution is the dynamically map the unmapped IPv4 addresses to some other honey pot instance. Thus, no IPv4 address is ever unmapped for more than a few seconds, unless all honey pots instances are in the state after a connection has been initiated towards them but have not yet been reinstantiated. We do the latter in AQS.

This choice requires that routable IPv4 addresses not be mapped statically to honey pots. Instead we assign each honey pot instance some fraction of routable IPv4 addresses assigned to AQS and the system dynamically balances the IPv4 address assignments as the IPv4 addresses are unmapped and mapped again.

In addition, if we assign the IPv4 addresses randomly this assures us that different honey pots systems are assigned the same IPv4 address at different times. This attribute is positive because many attackers scan the IPv4 address space sequentially or semi-sequentially. If routable IPv4 addresses were always mapped to the same private IPv4 addresses, the honey pot system mapped to the first routable IPv4 address in the honey pot server IPv4 address range would be disproportionately attacked.

Some network protocols do not work across a standard NAT device. These protocols usually encode the destination or source IPv4 address and/or TCP or UDP port numbers in the higher layers of the protocol. To get around this issue, NAT modules for some of these protocols have been developed. Since we want our honey pots to be as accessible as possible to a would be attacker or worm our NAT subsystem must support as many of these protocols as possible.

One known protocol that does not support NAT naturally and for which there is no current support in the Linux kernel, either in the standard kernel or in the form of a third-party module, is Microsoft's RPC protocol, which uses embedded source addresses. Microsoft DCOM uses Microsoft RPC. SMB (aka CIFS) and NetBIOS are known to work through a NAT device.

5.2.1.2 Implementation Details

Basics

The Linux kernel, via the *netfilter* project, includes NAT capabilities. As a result, the low-level details of NAT are already taken care of. The subsystem builds upon this base by managing the high-level mapping of routable IPv4 addresses to honey pot systems.

Routable Address States

Routable IPv4 addresses that are available to be mapped to a honey pot instance can be in one of three states: *Unassigned*, *Assigned*, and *Bound*.

Unassigned addresses are not mapped to any honey pot instance. They cannot be mapped to a honey pot instance until after a date and time recorded in the database. This is used to allow an address that goes from the *Bound* state to the *Unassigned* state to "cool off."

When an *Unassigned* address becomes available it may be mapped to a honey pot, along with a number of other addresses, at which point its state becomes *Assigned*.

When an *Assigned* address receives a packet it becomes *Bound*, while all other addresses mapped to the same honey pot instance become *Unassigned* and unmapped from the honey pot instance.

When a honey pot is suspended, the *Bound* address mapped to it is unmapped, and its state becomes *Unassigned*. The date and time when it becomes available is set in the future to give the address some time to cool off.

The state of routable addresses is maintained in the **HoneyPotAddresses** table.

Executables

A daemon, **dionaea-mapd**, maps a number of routable IPv4 addresses to honey pot instances. The daemon:

- Creates an exclusive lock so that only one instance of it executes at any one time.
- Loops and:
    - Looks up the status of all honey pot instances in the **HoneyPotInstances** table.
    - If the state of any honey pot instance changed to *Suspended*, *Halted*, *Analyzed* or *Reviewed* from the *Starting*, *Executing*, *Waiting*, *Contacted*, *Stopping*, or *Breached*, then for each such honey pot instance:
        - Looks up all the routable IPv4 addresses mapped to the honey pot instance in the **HoneyPotAddresses** and **HoneyPotInstances** table.
        - Removes all rules from the *mangle* table's *PREROUTING* chain that match as the IPv4 destination address one of the routable IPv4 addresses mapped to the honey pot instance.

- Removes all rules from the *nat* table's *PREROUTING* chain that match as the destination IPv4 address one of the routable IPv4 addresses mapped to the honey pot instance.
- Looks up the private IPv4 addresses of the honey pot system in the **HoneyPotSystems** table.
- Removes all rules from the *nat* table's *POSTROUTING* chain that matches as the source IPv4 address the honey pot system's private IPv4 address.
- Sets the state of the any addresses mapped to the honey pot instance to *Unassigned* in the **HoneyPotAddresses** table. If the address had a state of *Bound* the time when it becomes available should be set to ten minutes in the future (cooling off period), so that if the attacker attempts to connect the honey pot again within ten minutes he won't attack a new instance of it, or an instance of another honey pot.
- Insert a record in the **HoneyPotAddressEvents** table to indicate the state of *Unassigned* for any addresses mapped to the honey pot instance.

o Looks up the status of all addresses in the **HoneyPotAddresses** table.

o If there is a new honey pot instance with a state of *Executing*, or if the status of any address changed to *Unassigned* from some other state:

- Removes all rules from the *mangle* table's *PREROUTING* chain that match as the incoming interface the interface connected to the Internet and a state of *NEW* with a target of *QUEUE*.
- Computes the percentage of addresses that should be assigned to each honey pot instance in the *Executng* or *Waiting* state.
- For each address in the *Assigned* state, or in the *Unassigned* state whose availability date and time has passed:
  - Assigns, or reassign, it to a honey pot instance, selecting the instance by a random weighted selection in correspondence with the percentage of addresses that should be assigned to each honey pot instance, changing the state of the address in the **HoneyPotAddresses** table accordingly.
  - Insert a record in the **HoneyPotAddressEvents** table to indicate the status of *Assigned*.
  - Inserts a rule in the *mangle* table's *PREROUTING* chain that matches as the incoming interface the interface connected to the Internet, a state of *NEW*, and an IPv4 destination address equal to the routable IPv4 address

assigned to the honey pot instance, with a target of *QUEUE*.

- For new honey pot instance with a status of *Executing*:
  - Modifies the record of the honey pot instances in the **HoneyPotInstances** table, setting the status to *Waiting*.
  - Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance's status is now *Waiting*.
- Sleep for 5 seconds.

- Removes the exclusive execution lock.

A daemon, **dionaea-natd**:

- Waits for the kernel to pass it a packet that matches one of the rules in the *mangle* table's *PREROUTING* chain with a target of *QUEUE*, signaling the initiation of a connection towards the honey pot instance.
- Looks up the honey pot instance associated with the IPv4 destination address of the packet in the **HoneyPotAddresses** and **HoneyPotInstances** tables.
- Looks up all the routable IPv4 addresses mapped to the honey pot instance in the **HoneyPotAddresses** table.
- Removes all rules from the *mangle* table's *PREROUTING* chain that match as the IPv4 destination address one of the routable IPv4 addresses mapped to the honey pot instance.
- For each routable IPv4 address mapped to the honey pot instance, except the one that is the destination of the packet received:
  - Sets their state to *Unassigned* in the **HoneyPotAddresses** table.
  - Inserts a record in the **HoneyPotAddressEvents** table to indicate the status of *Unassigned*.
- Looks up the honey pot instance private IPv4 address in the **HoneyPotSystems** table.
- Insert a new rule in the *nat* table's *PREROUTING* chain that matches as the incoming interface the interface connected to the Internet with a target of *DNAT* that maps the destination IPv4 address from the received packet to the honey pot instance's private IPv4 address.
- Insert a new rule in the *nat* table's *POSTROUTING* chain that matches as the outgoing interface the interface connected to the Internet with a target of *SNAT* that maps the honey pot instance's private IPv4 address to the source IPv4 address from the received packet.
- Sets the state of the destination IPv4 address from the received packet to *Bound* in the **HoneyPotAddresses** table.

- Inserts a record in the **HoneyPotAddressEvents** table to indicate the status of *Bound.*
- Sets the status of the honey pot instance to *Contacted* in the **HoneyPotInstances** table.
- Inserts a record to the **HoneyPotInstancesEvents** table to indicate the honey pot instance status is now *Contacted.*
- Responds to the kernel with a verdict of *NF_ACCEPT* for the packet, which will cause the packet continue its route through the kernel NAT, connection tracking and packet filtering code.

The daemons can modify the kernel's *nat* table via the *libiptc* C library and the *IPTables* Perl module.

The **dionaea-natd** daemon can monitors network traffic for the first packet to initiate a connection to a honey pot instance via the *libipq* C library and the *perlipq* Perl module.

The following connection tracking/NAT modules must be configured in the Linux kernel: *ftp, eggdrop-conntrak, irc-conntrack, pptp, record-rpc, snmp-nat, talk-conntrack-nat,* and *tftp.*

5.3 Honey Pot Support Services Server

The honey pot support services server hosts the following subsystem:

- Honey Pot Support Services, such as DHCP and DNS services, which assist in the correct functioning of the honey pots.

5.3.1 Honey Pot Support Services

The Honey Pot Support Services subsystem provides network services to the honey pot systems that are necessary for their proper functioning.

The services include:

- DHCP.
- Recursive DNS.

5.3.1.1 Reasoning

DHCP Service

We do not wish to need to manually configure each honey pot archetype. We want to be able to create a honey pot archetype and use it to instantiate as many honey pot systems of the type represented by the archetype as necessary. DHCP permits the honey pot instance to learn its IPv4 address, subnet mask, gateway address, and DNS server without them being hard-coded into the configuration stored in the archetype.

Recursive DNS Service

The Domain Name System maps alphanumeric hostnames to IPv4 addresses. If some malicious code or attacker breaches a honey pot instance and attempts to make a network connection to some system using a hostname, the hostname needs to be mapped to an IPv4 address by a DNS server. Without one the network connection will fail, the attacker may detect something is wrong, and we will fail to find out why the malicious code or attacker wished to connect to the remote system.

5.3.1.2 Implementation Details

DHCP Service

When it comes to DHCP under Linux/UNIX the ISC DHCP server is the only game in town. We use ISC DHCP version 3.0 with the paranoia patch (http://www.episec.com/people/edelkind/patches/dhcp/dhcp-3.0+paranoia.patch), which adds support for changing the root directory of the server and executing the server as a non-root user and group.

The DHCP sever executes with a different root directory, a non-root user id, and non-root group id.

The DHCP server is configured to assign each honey pot system a static IPv4 address based on its Ethernet address. When a new honey pot system is added to the honey pot server it is assigned an Ethernet address, which is stored in the database. At the same time, the DHCP server's configuration file must be modified to add the new honey pot system's Ethernet address and its matching static private IPv4 address.

The lease timeout is set to 86400 seconds (one day) to minimize the DHCP network traffic.

Recursive DNS Service

For the recursive DNS server we are using the *dnscache* tool from D.J. Bernstein's *dbjdns* software. The tool runs under its own user and group ids, and executed with a different root directory in its standard installation.

The *dnscache* tool must be configured to only accept DNS queries from the 192.68.0.0/16 honey pot network by creating the empty file */etc/dnscache/root/ip/192.68*.

5.4 Honey Pot Management Server

The honey pot management server hosts the following subsystems:

- Post-Intrusion Automated Analysis, which analyzes a honey pot's state after an intrusion; analyzes the captured network traffic to detect attacks or probes, determine the operating system of the attacker, and decrypt some network traffic; requests probes of attacking system; and submits executable samples to DIS.
- Web-based Interface.

5.4.1 Post-Intrusion Automated Analysis

The Post-Intrusion Automated Analysis subsystem performs some automated analysis of the intrusion, feeds samples to DIS, and makes information available to the analysis via the database.

The subsystem:

- Determines whether any changes where made to the honey pot instance's file system, including detecting new files.
- Attempts to determine the type of any new files discovered in the honey pot instance's file system.
- Attempts to determine whether any new files found on the honey pot instance are an instance of some known malicious code by querying the Malware Oracle.
- Analyzes the captured network traffic for probes and attacks (NIDS).
- Performs passive network stack fingerprinting to determine the platform of the IPv4 addresses communicating with the honey pot instance.
- Decrypts HTTP over SSL/TLS (HTTPS) network traffic.
- Probes remote hosts.
- Submits any new executable files to DIS.
- Makes the breach information available to the analysts via the database.

5.4.1.1 Reasoning

Performing an analysis of a computer intrusion is a resource intensive task. By automating some of the most common analysis processes we alleviate the load on the analysts and allow them to concentrate in other parts of the analysis process. The automated analysis also permits an analysis make a quick determination whether the incidents is the result of some known malicious code or was conducted with a known attack, allowing them to concentrate in the more interesting braches.

Detecting File System Changes

VMware under Linux supports the mounting of virtual disks. Since Linux supports a number of file systems this permits us to access a honey pot's file system, although through the lens of the POSIX API. This facility permits us to perform a basic file integrity analysis of the honey pot's file system to detect some file changes, such a changes in content, deletion of files, and addition of files. Because some of the file attributes available in some non-native Linux file systems do not map well to the standard UNIX file system semantics, things such as NTFS ACLs and streams, as well as FAT attributes, cannot be analyzed easily. Nonetheless, the basic analysis is capable of detected most of the common changes to a file system during a breach.

There are a number of tools readily available that perform file system integrity verification. They will serve us well in detecting changes to the virtual disks mounted as directories under Linux. In some future revision of AQS we may need to augment these

tools or write our own if we decide to attempt to capture some of the file system attributes that are not expressed via the common POSIX API.

Detecting the Type of a File

Guessing the file type of modified or created files in the breached honey pot instance's file system can help the analyst prioritize his analysis work by steering him to executable files that may contain malicious code, exploits, or other hacking tools.

Detecting Known Malicious Code

Detecting whether any of the modified or created files in the breached honey pot instance's file system are an instance of some known malicious code helps the analyst focus on his analysis in files of unknown contents or skip performing the analysis of some breaches altogether.

Detecting Network Probes and Attacks

Detecting known network probes and attacks can help the analysts determine whether the attack pattern resembles that of known malicious code or tools, and whether the attack was known or unknown, allowing them to best determine what to analyze.

Detecting the Attacker's Platform

Detecting the operating system of the source of any probes or attacks can help the analysts distinguish between common attack patterns (e.g. attacks associated with CodeRed being launched from a Windows host) and unusual ones (e.g. attacks associated with CodeRed being launched from a Solaris host), thus helping them decide what to analyze and what to ignore.

Decrypting SSL/TLS Network Traffic

Network encryption can hide the details of an attack in the analysis of captured network traffic. In particular, malicious code or attackers may use HTTP over SSL/TLS (HTTPS) to hide their attack from network monitoring devices, such as intrusion detection systems. It would be useful to be capable to decrypt such traffic to make analysis easier.

5.4.1.2 Implementation Details

5.4.1.2.1 Detecting File System Changes

The *AIDE* file system integrity verification tool -- http://www.cs.tut.fi/~rammer/aide.html -- allows its user to detect file system changes. We use it to detect file system changes in honey pot instances.

When a new honey pot virtual disk archetype is created a corresponding *AIDE* configuration file and database must be created. These configuration file and database are used to detect any changes to the honey pot instance's file system. The archetype *AIDE* configuration file is stored under **/var/dionaea/archetypes/<*ar*>/aide/aide.conf**. The archetype *AIDE* database is stored under **/var/dionaea/archetypes/<*ar*>/aide.db**.

The *AIDE* configuration file includes a file of file and directory to check for changes or to ignore. This list must be finely tuned when creating an archetype so that files that change during the normal executing of the system are not included. All selection lines must also

start the file to check or ignore with **/var/dionaea/mount**. This is the directory used to mount any file system for analysis.

When we perform post-intrusion analysis on a honey pot instance we compare the archetype *AIDE* database to the post-intrusion file system by executing the *aide* command with arguments of **–C,–config=/var/dionaea/archetypes/<*ar*>/aide/aide.conf, –B "database=/var/dionaea/archetypes/<*ar*>/aide/aide.db"**, and **–r file:/var/dionaea/post/<*id*>-<*instance*>/analysis/aide.out**.

The output of *AIDE* can be machine parsable, although it was not specifically designed for this purpose. While the analyst can make use of the output as it is, we also parse the output to determine what files were modified or a new, to record these changes in the database.

The output of *AIDE* is used to determine if any files have been modified or created in the breached honey pot instance. For the purpose of capturing sample we are only interested in modified files whose checksum has changed, or new files.

Files that have been modified with a new checksum or have been created are copied to the **/var/dionaea/post/<*id*>-<*instance*>/analysis/files/samples** directory located with the rest of the breached honey pot instance's state. As they are copied the files are renamed so as to include their pathname within their filename. For instance, if the file **WINNT/REGEDT.EXE** has been modified or created it is copied under the filename **WINNT=REGEDT.EXE**. This insures that two files with the same filename but in different locations within the file system of the honey pot are assigned unique names when copied.

Attributes of the new or modified file are stored in a file with the same name in the **/var/dionaea/post/<*id*>-<*instance*>/analysis/files/info** directory. The attributes that are saved include:

- The path name of the sample, including the drive letter for platforms with that concept.
- The filename.
- Its size.
- Its timestamps.
- Its MD5 and SHA1 hash.
- The platform the sample was captured on.

The list of deleted, modified, or new files is stored in the database in records associated with this breach.

The command **dionaea-post-aide** takes as arguments the honey pot system id and honey pot instance id, and:

- Lookups up the honey pot system's archetype in the **HoneyPotSystems**, **VMwareHoneyPotSystems**, and **VMwareHoneyPotArchetypes** tables.

- Executes the *aide* command with arguments of **–C, –config=/var/dionaea/archetypes/<*ar*>/aide/aide.conf, –B "database=/var/dionaea/archetypes/<*ar*>/aide/aide.db"**, and **–r file:/var/dionaea/post/<*id*>-<*instance*>/analysis/aide.out**
- Parses the **/var/dionaea/post/tmp/<*id*>-<*instance*>/analysis/aide.out** file in search of modified files with a new checksum or new files.
- Inserts the data into the **FileSystemChanges** table.

5.4.1.2.2 Detecting the Type of a File

The Perl module *File::Type* -- http://www.perl.com/language/ppt/src/file/file.slay.html -- is capable of estimating a file type. The latest version of the standard *file* command can be found at ftp://ftp.gw.com/mirrors/pub/unix/file/. While neither of these tools can detect all file types they provide some information for an analyst to prioritize their analysis.

If any files in the honey pot instance's file system were modified or new ones created, then each one of them will be passed through one of these tools to estimate its file type. The estimated type of the modified or new files is stored in the database.

The command **dionaea-post-ftype** takes as an argument a file to analyst and:

- It estimates the file type by using the Perl module *File::Type* or the *file* command.
- Returns the file type.

5.4.1.2.3 Detecting Known Malicious Code

The Malware Oracle subsystem is used to determine whether any of the modified or new files in the breached honey pot instance's file system are an instance of some known malicious code.

If any files in the honey pot instance's file system were modified or new ones creates, then each one of them is submitted to the Malware Oracle for analysis. The answer from the Malware Oracle is stored in the database.

The interface to the Malware Oracle is specified in the DeepSight Malware Oracle 2.0 Functional Requirements Document.

5.4.1.2.4 Detecting Network Probes and Attacks

The *Snort* IDS can examine network traffic and generate alerts if it detects probes or attacks. It also has the capability of examining captured network traffic stored in a file in the *pcap* dump file format.

The probe and attack signatures used by *Snort*, or rules, as well as its configuration file, are stored in the **/var/dionaea/conf/snort** directory. *Snort* is configured to reassemble IP packets (*frag2* preprocessor), TCP streams (*stream4* & *stream4_reassemble* preprocessors), decode HTTP (*http_decode* preprocessor), RPC (*rpc_decode*

preprocessor), and TELNET (*telnet_decode* preprocessor) protocols, and detect port scans (*portscan* preprocessor).

Snort is also configured to use the *alert_fast* output plug-in with an output filename of **snort.txt**, and the *xml* output plug-in with an output filename of **snort.xml**. These files are created in the **/var/dionaea/post/tmp/<*id*>-<*instance*>/analysis** directory.

The **dionaea-post-snort** command takes an as arguments the honey pot system id and the honey pot instance id. It:

- Executes the *snort* command with arguments of **–NqUyz**, **-k all**, **–l /var/dionaea/post/tmp/<*id*>-<*instance*>/analysis**, **–c /var/dionaea/conf/snort/snort.conf**, and **–r /var/dionaea/post/tmp/<*id*>-<*instance*>/netcap**.
- Parses the **/var/dionaea/post/tmp/<*id*>-<*instance*>/analysis/snort.xml** file and inserts the data into the **NetworkEvents** table.

5.4.1.2.5 Detecting the Attacker's Platform

The passive OS fingerprinting tool (*p0f*) can examine network traffic and estimate the operating system executing in the remote hosts by matching the parameters of the packets they produce with known fingerprints for some platforms. It also has the capability of examining captured network traffic stored in a file in the *pcap* dump file format.

P0f takes an input a fingerprint information file that contains information of identifying operating systems based on their network traffic. This file is stored in **/var/dionaea/conf/p0f-fingerprints**.

P0f is executed to examine the captured network traffic to/from the honey pot instance. Since the NAT code could modify some of the network traffic we examine the network traffic captured before NAT is applied, which is stored in the file **/var/dionaea/post/<*id*>-<*instance*>/netcap**. Its output is stored in **/var/dionaea/post/<*id*>-<*instance*>/analysis/p0f.out**.

The command **dionaea-post-p0f** takes as arguments the honey pot system id and honey pot instance id, and:

- Executes the *p0f* command with arguments of **–t**, **-q**, **–f /var/dionaea/conf/p0f-fingerprints**, **-s /var/dionaea/post/tmp/<*id*>-<*instance*>/netcap**, **-o /var/dionaea/post/tmp/<*id*>-<*instance*>/analysis/p0f.out**.
- Parses the file **/var/dionaea/post/tmp/<*id*>-<*instance*>/analysis/p0f.out** and inserts the data in the **RemoteSystems** table. This not only determines the remote system platform, but also builds the list of remote systems.

5.4.1.2.6 Decrypting SSL/TLS Network Traffic

The *ssldump* tools is a SSL/TLS traffic analyzer that is capable of decrypting any data transmitted through SSL/TLS if provides with the private RSA key of the SSL/TLS server.

Since *ssldump* requires the private RSA key of the SSL/TLS server we generate one key pair that we configure as the server's key in any honey pot system configured to run an HTTP server. The private key is stored in **/var/dionaea/conf/ssldump/key.pem**. The public key certificate is stored in **/var/dionaea/conf/ssldump/cert.pem**.

To generate the key and a self-signed X.509 certificate we use the *openssl* command as follow:

```
openssl req -new -x509 -days 999 -nodes -out cert.pem \
        -keyout key.pem
```

5.4.1.2.7 Probing Remote Hosts

The Inquisitor system provides a service through which remote hosts can be probed to actively determine their operating system and the status of their TCP and UDP port number, as well as some other information.

The command **dionaea-post-probe** takes as an argument the remote system id probe. It:

- Looks up the remote system's IPv4 address in the **RemoteSystems** table.
- Contacts the Inquisitor server.
- Commands it to probe the IPv4 address.
- Reads the request id returned by the Inquisitor server.
- Creates a file named as request id returned by the Inquisitor server whose contents are the remote system's id in the directory **/var/dionaea/inquisitor/outstanding**.

A daemon, **dionaea-post-probed**, pools the Inquisitor server to collect the probe results. It:

- Writes its process id to the file **/var/dionaea/inquisitor/pid**.
- Loops and:
    - Reads the directory **/var/dionaea/inquisitor/outstanding**.
    - For each file it:
        - Probes the Inquisitor server to determine whether the probing has completed.
        - If it has:
            - Reads the probe result.
            - Read the file's content, which has the remote system id.
            - Parses the XML result.
            - Inserts the result into the **RemoteSystems** and **RemoteSystemPorts** tables.
            - Remote the file from the directory **/var/dionaea/inquisitor/outstanding**.

- Sleeps for 5 minutes.

5.4.1.2.8 Submitting Samples to DIS

Any modified or new executable files found on the honey pot instance's file system are submitted to DIS for analysis. The protocol to communicate with DIS is documented in *Immune System network protocol, twelfth draft.*

A daemon, **dionaea-disd**, monitors the directory that contains the new or modified files found in the honey pot instances for changes and submits any files added to it to a DIS gateway over the HTTP protocol via a POST command to the */AVIS/postSuspectSample* URI. At the same time the file sample's attributes are transmitted via HTTP headers. (See *Immune System network protocol, twelfth draft*, Section 10.4.) It then moves the submitted file to a pending directory.

The daemon does not need to send a *X-Sample-Category*, *X-Customer-Credential*, or *X-Platform-Correlator* HTTP headers. They are not currently in use by DIS.

The daemon uses a value of *900* in the *X-Sample-Priority* HTTP header when submitting a sample.

The daemon uses the value *unknown* in the *X-Sample-Reason* HTTP header when submitting a sample.

Another daemon periodically polls the DIS gateway to determine the status of file samples submitted earlier that have not reached a terminal state. It does this by issuing an HTTP HEAD command for the */AVIS/getSampleStatus* partial URI to the DIS gateway. (See *Immune System network protocol, twelfth draft*, Section 10.7.) One the gateway returns a status for the submitted sample the status is stored in the database. It also moves the file from the pending directory to a done directory. The subsystem stops querying the status of a submitted sample once its status is a terminal state. The *X-Date-Analyzed* header in a response indicates a terminal state. (See *Immune System network protocol, twelfth draft*, Section 8.2.)

Also stored in the database is a human friendly status message. This message is mapped from the returned status to a standard message for most statuses, mapped to the contents of the returned *X-Attention* header for a status of *attention*, and mapped to the contents of the returned *X-Error* header for a status of *error*, which in turn is mapped to a human friendly error message stored in the database (See *Immune System network protocol, twelfth draft*, Section 8.5.)

5.4.1.2.9 Putting It All Together

A daemon, **dionaea-post-analysisd**, drives the post-intrusion analysis of the breached honey pot instance. The daemon:

- Loops and:
  - List the file in the directory **/var/dionaea/post/incoming**.
  - If it finds any files in the directory:

- For any files of the form ***<id>-<instance>*.tar** that it finds, it:
  - Extracts the archive to the directory **/var/dionaea/post/tmp/*<id>-<instance>***.
  - Executes **dionaea-post-analysis** with the directory as an argument.
  - Moves the directory **/var/dionaea/post/tmp/*<id>-<instance>*** to **/var/dionaea/post/done/*<id>-<instance>***.
  - Modifies the record of the honey pot instances in the **HoneyPotInstances** table, setting the status to *Analyzed*.
  - Inserts a record to the **HoneyPotInstanceEvents** table to indicate the honey pot instance state is now *Analyzed*.
- Else, it sleeps for 30 seconds.

A command, **dionaea-post-analysis**, does the post-intrusion analysis of the breached honey pot instance. The command:

- Executes the **dionaea-post-p0f** command with arguments of the honey pot system id and the honey pot instance id.
- Executes the **dionaea-post-snort** command with arguments of the honey pot system id and the honey pot instance id.
- Executes the ssldump command with arguments of **–dnN**, **–r /var/dionaea/post/tmp/*<id>-<instance>*/netcap**, and **–k /var/dionaea/conf/ssldump/cert.pem**, while redirecting its output to the file **/var/dionaea/post/tmp/*<id>-<instance>*/analysis/ssldump.out**.
- Mounts the honey pot instance's virtual disk in the directory **/var/dionaea/mount** in read-only mode by using the **vmware-mount.pl** command.
- Executes the **dionaea-post-aide** command with arguments of the honey pot system id and the honey pot instance id.
- If any modified files with new checksums or new files are found:
  - It copies the file to the **/var/dionaea/post/tmp/*<id>-<instance>*/analysis/files/samples** directory while at the same time renaming it so that its pathname is encoded in the new filename (e.g. **WINNT/REGEDT.EXE** becomes **WINNT=REGEDT.EXE**).
  - It estimates the file type by using **dionaea-post-ftype** command and updates the record in the **FileSystemChanges** table.
  - It writes to a file with the same name but in the directory **/var/dionaea/post/tmp/*<id>-<instance>*/analysis/files/info** the following attributes of the copied file:

- The path name of the sample, including the drive letter for platforms with that concept.
- The filename.
- Its size.
- Its timestamps.
- Its MD5 and SHA1 hash.
- The platform the sample was captured on.
- The estimated file type.

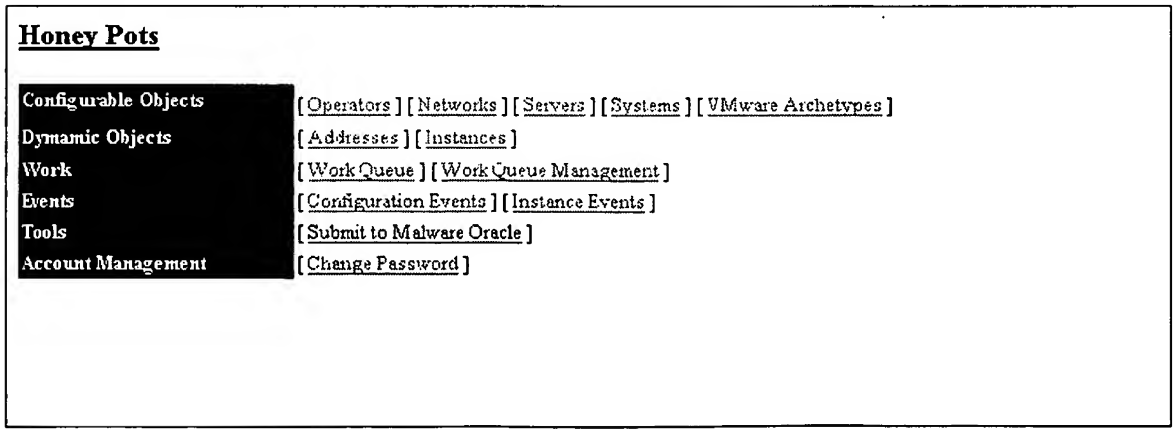
- Umounts the honey pot instance's virtual disk mount at **/var/dionaea/mount/**.
- For each remote system in the **RemoteSystems** table associated with this instance it:
  - Executes **dionaea-probe** with the remote system id.
- Submits any modified or new files to the Malware Oracle.
- Submits any modified or new *executable* files to DIS.

5.4.2 Web-Interface

The web interface provides a mechanism for operators to interact with the AQS.

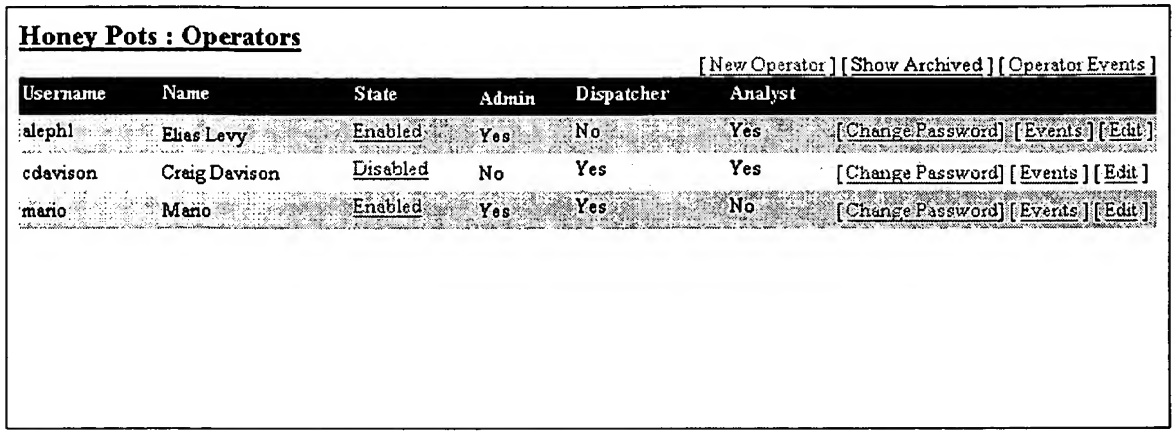
5.4.2.1 Home Page

This is the home page of the AQS web interface. The example below shows the page displayed to an operator with roles of administrator, dispatcher, and analyst. The rows Configurable Objects, and Events are only displayed to administrators. The Work Queue link is only displayed to analysts. The Work Queue Management link is only displayed to dispatchers.



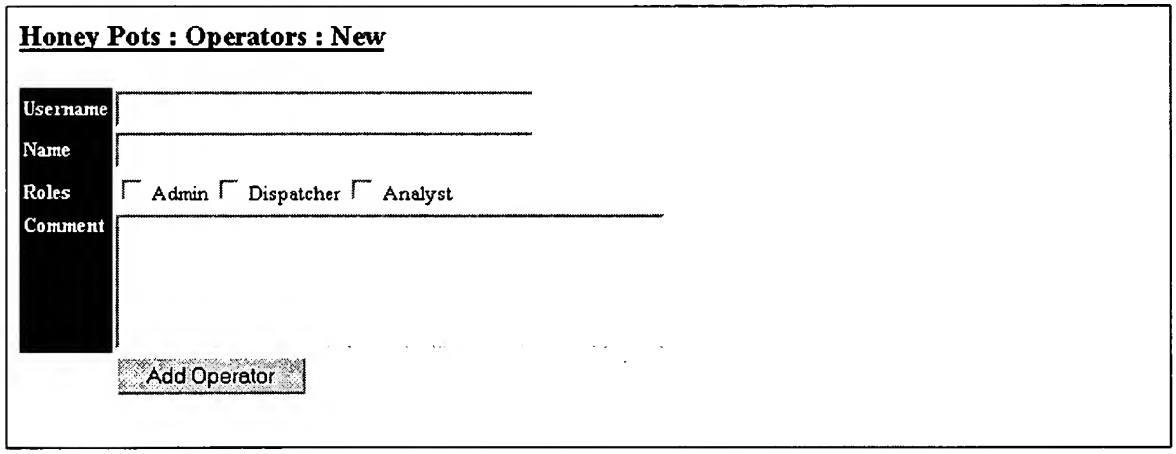
5.4.2.2 Honey Pots : Operators

This page displays a list of operator accounts configured in the system and permits and administrator to create new accounts, edit existing accounts, change passwords, and display configuration events associated with an operator object.



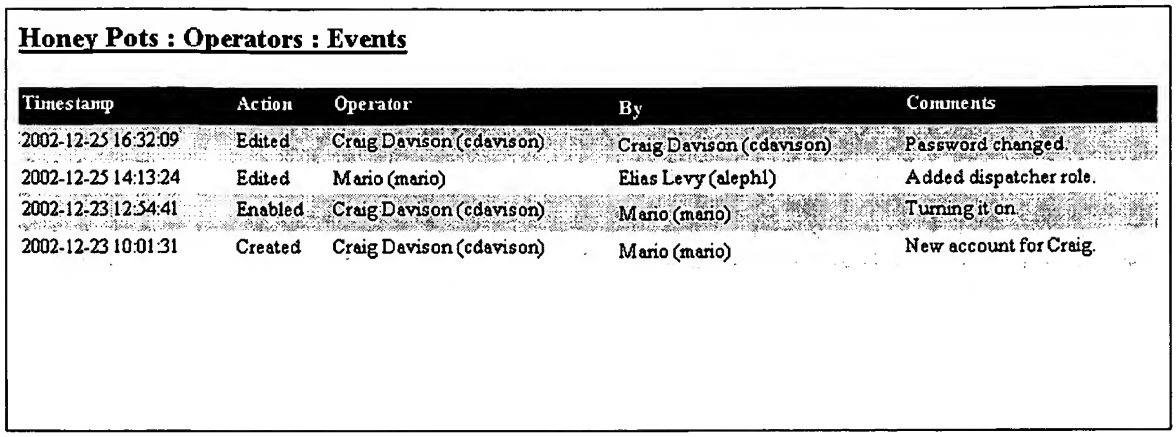
5.4.2.3 Honey Pots : Operators : New

This page permits an administrator create a new operator account.



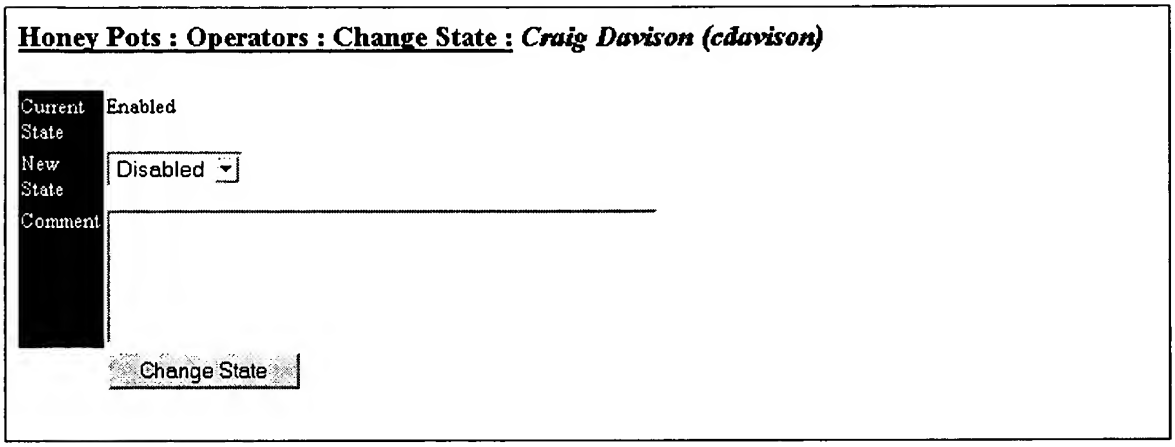
5.4.2.4 Honey Pots : Operators : Events

This page displays a list of all events affecting operator objects.



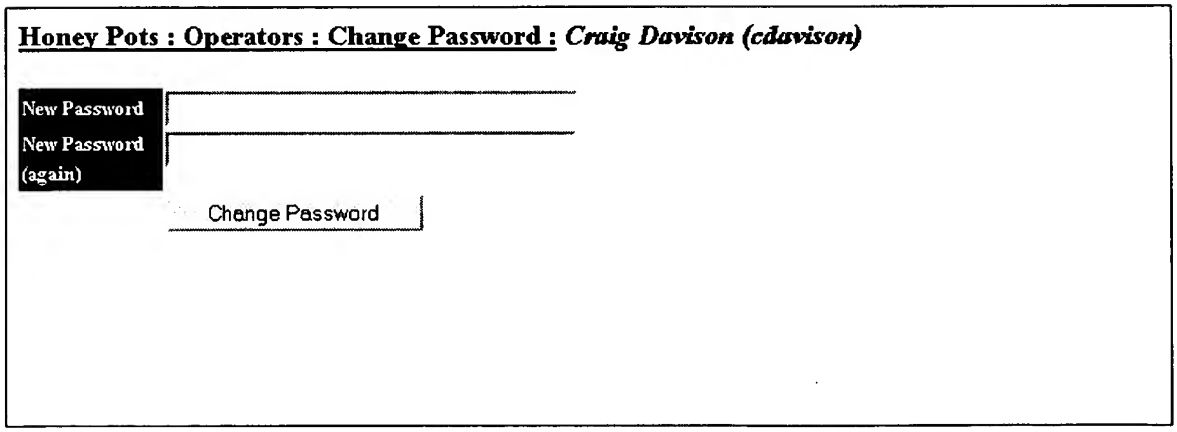
5.4.2.5 Honey Pots : Operators : Change State : <Operator>

This page permits an administrator change the state of an operator object.



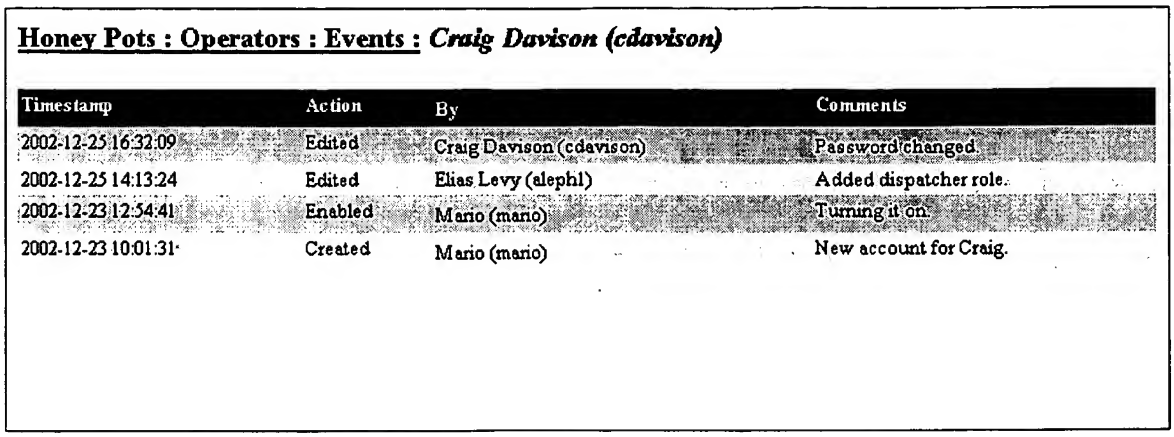
5.4.2.6 Honey Pots : Operators : Change Password : <Operator>

This page permits an administrator change an operator's password.



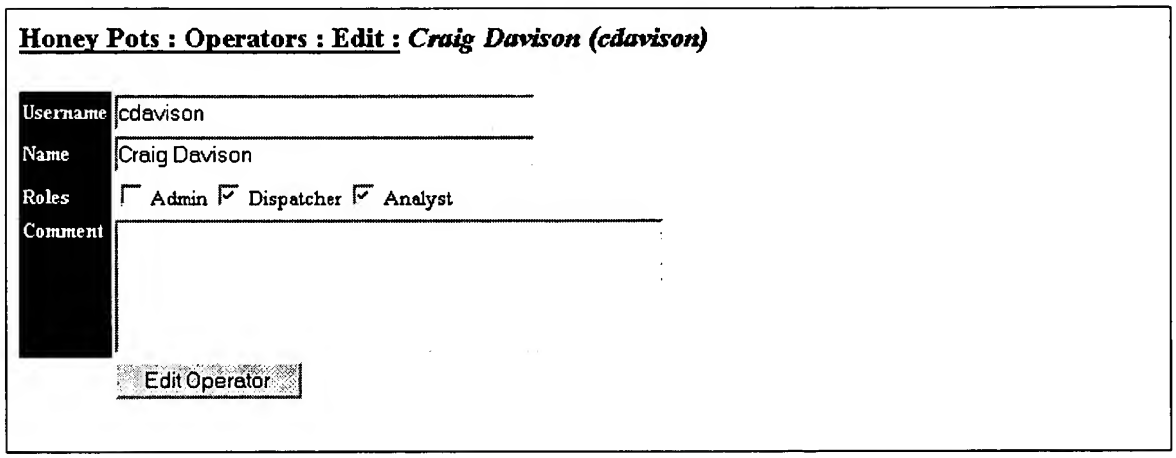
5.4.2.7 Honey Pots : Operators : Events : <Operator>

This page permits an administrator view the events affecting a specific operator object.



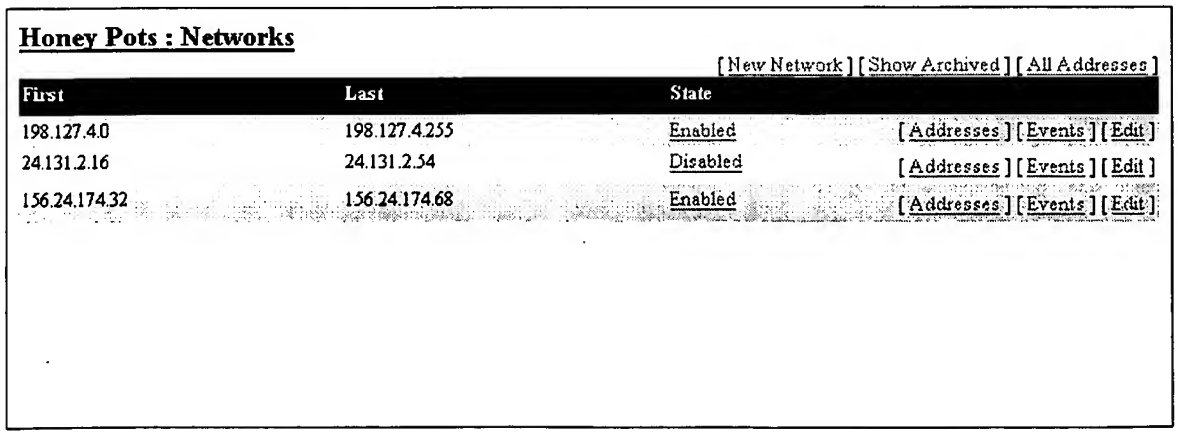
5.4.2.8 Honey Pots : Operators : Edit : <Operator>

This page permits an administrator edit an operator object.



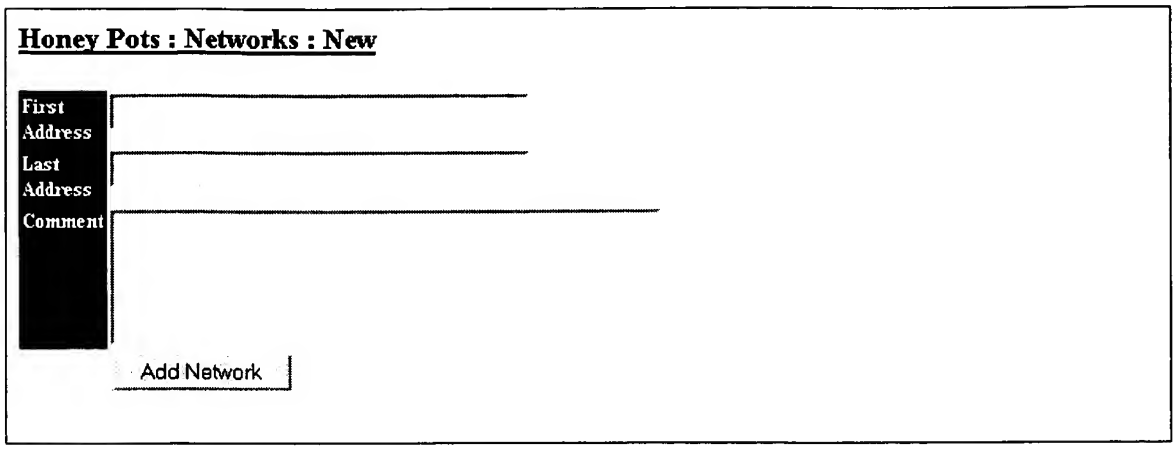
5.4.2.9 Honey Pots : Networks

This page displays a list of honey pot networks configured in the system and permits and administrator to configure a new network, edit existing networks, show the state of their associated addresses, and display configuration events associated with a network object.



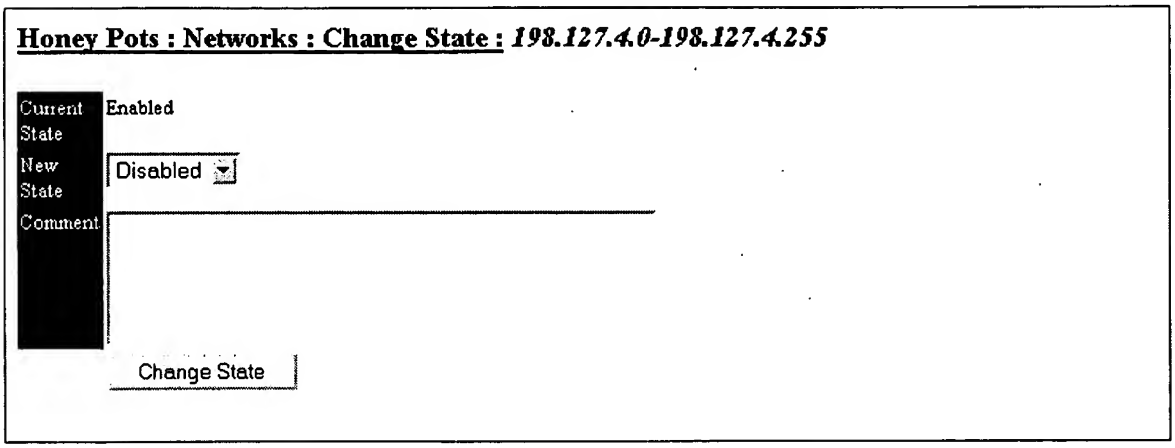
5.4.2.10 Honey Pots : Networks : New

This page permits the administrator create a new network object.



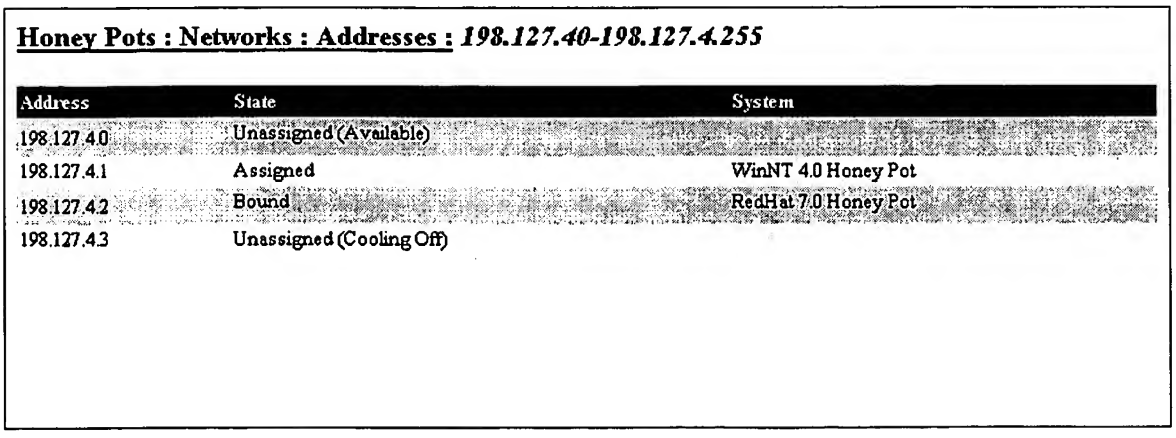
5.4.2.11 Honey Pots : Networks : Change State : <Network>

This page permits the administrator modify the state of a network object.



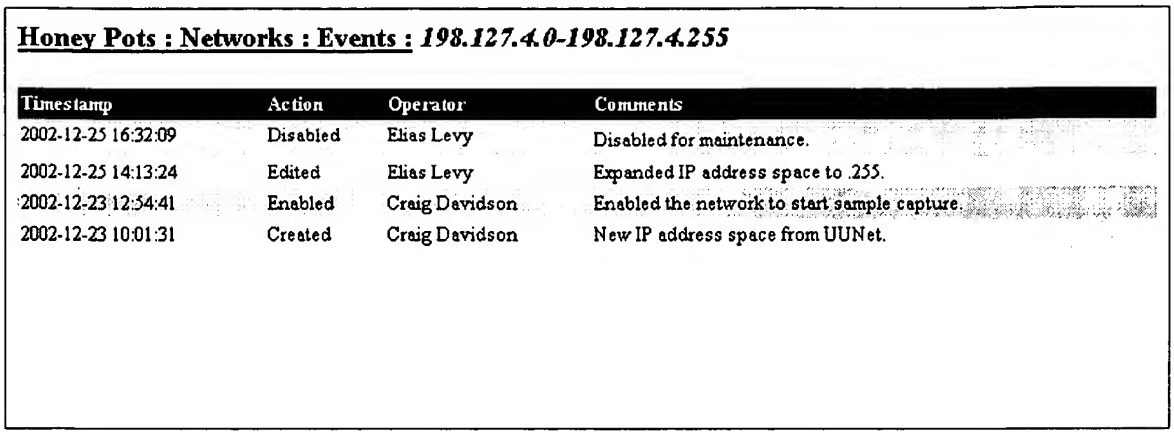
5.4.2.12 Honey Pots : Networks : Addresses : <Network>

This page permits the administrator determine the status of IPv4 addresses within the network.



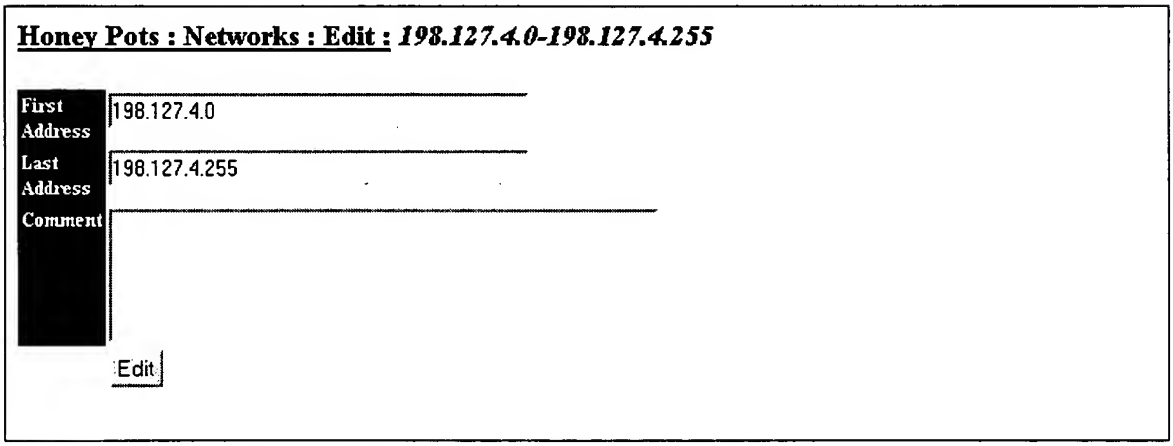
5.4.2.13 Honey Pots : Networks : Events : <Network>

This page permits the administrator view configuration events associated with a network object.



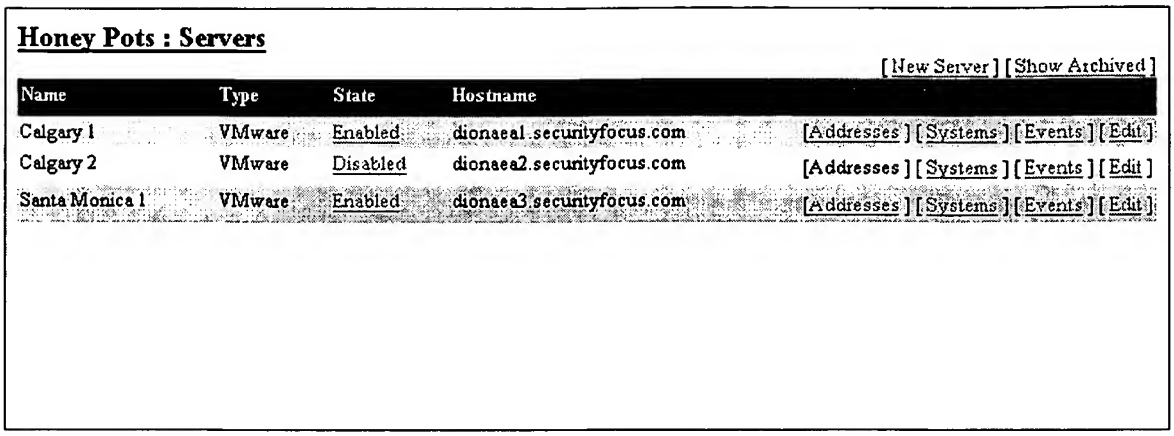
5.4.2.14 Honey Pots : Networks : Edit : <Network>

This page permits the administrator to edit a network object.



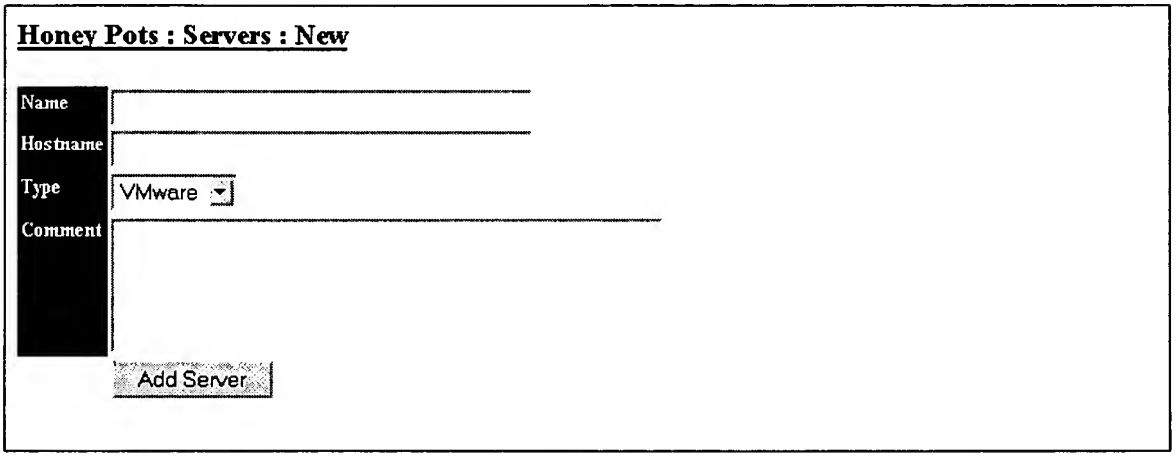
5.4.2.15 Honey Pots : Servers

This page displays a list of honey pot servers objects configured in the system and permits the administrator to create new honey pot server objects, edit them, display their honey pot systems, or view the events associated with them.



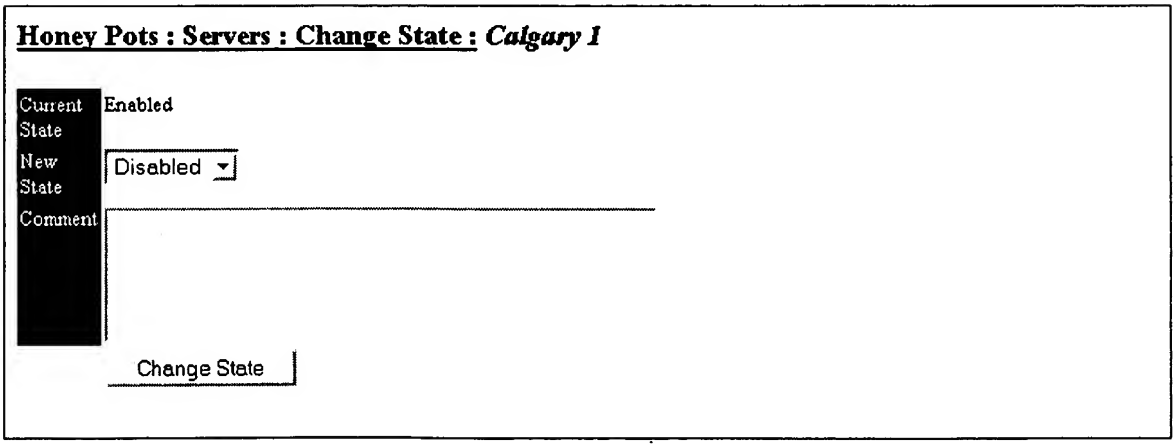
5.4.2.16 Honey Pots : Servers : New

This page permits the administrator create a new honey pot server object.



5.4.2.17 Honey Pots : Servers : Change State : <Server>

This page permits the administrator to change the state of a honey pot server object.



5.4.2.18 Honey Pots : Servers : Addresses : <Server>

This page permits the administrator view the IPv4 addresses assigned or bound to a honey pot server object.

Honey Pots : Servers : Addresses : *Calgary 1*

Address	State	System
198.127.4.0	Assigned	WinNT 4.0 Honey Pot
24.131.2.32	Assigned	WinNT 4.0 Honey Pot
24.131.2.33	Bound	RedHat 7.0 Honey Pot

5.4.2.19 Honey Pots : Servers : Systems : <Server>

This page permits the administrator view the honey pot systems associated with a honey pot server object, to create a new honey pot server object, edit an existing one, or view the configuration events associated with one.

Honey Pots : Servers : Systems : *Calgary 1*

[New System] [Show Archived]

Name	State	Weight	Interface	Private Address	Archetype	Ethernet Address	
Windows NT 4.0	Enabled	10	vmware0	198.169.0.2	Windows NT 4.0 Default Install	00:50:56:00:00:01	[Instances] [Events] [Edit]
Windows 2000	Disabled	25	vmware1	198.169.1.2	Windows 2000 Default Install	00:50:56:00:00:02	[Instances] [Events] [Edit]
Windows 2000 SP3	Enabled	35	vmware2	198.169.2.2	Windows 2000 SP3 Default Install	00:50:56:00:00:03	[Instances] [Events] [Edit]
RedHat 7.0	Enabled	30	vmware3	198.169.3.2	RedHat 7.0 Default Install	00:50:56:00:00:04	[Instances] [Events] [Edit]

5.4.2.20 Honey Pots : Servers : Events : <Server>

This page permits the administrator view configuration events associates with a honey pot server object.

Honey Pots : Servers : Events : *Calgary 1*

Timestamp	Action	Operator	Comments
2002-12-25 16:32:09	Enabled	Elias Levy	Maintenance completed.
2002-12-25 14:13:24	Disabled	Elias Levy	Disabled the server for maintenance.
2002-12-23 12:54:41	Enabled	Craig Davidson	Enabled the server to start sample capture.
2002-12-23 10:01:31	Created	Craig Davidson	Added new calgary honey pot server.

5.4.2.21 Honey Pots : Servers : Edit : <Server>

This page permits the administrator edit a honey pot server object.

Honey Pots : Servers : Edit : *Calgary 1*

Name	Calgary 1
Hostname	dionaea1.securityfocus.com
Type	VMware
Comment	

Edit

5.4.2.22 Honey Pots : Systems

This page permits the administrator view all honey pot system objects, create new ones, edit existing ones, view the instance objects associated with one, or the events associated with one.

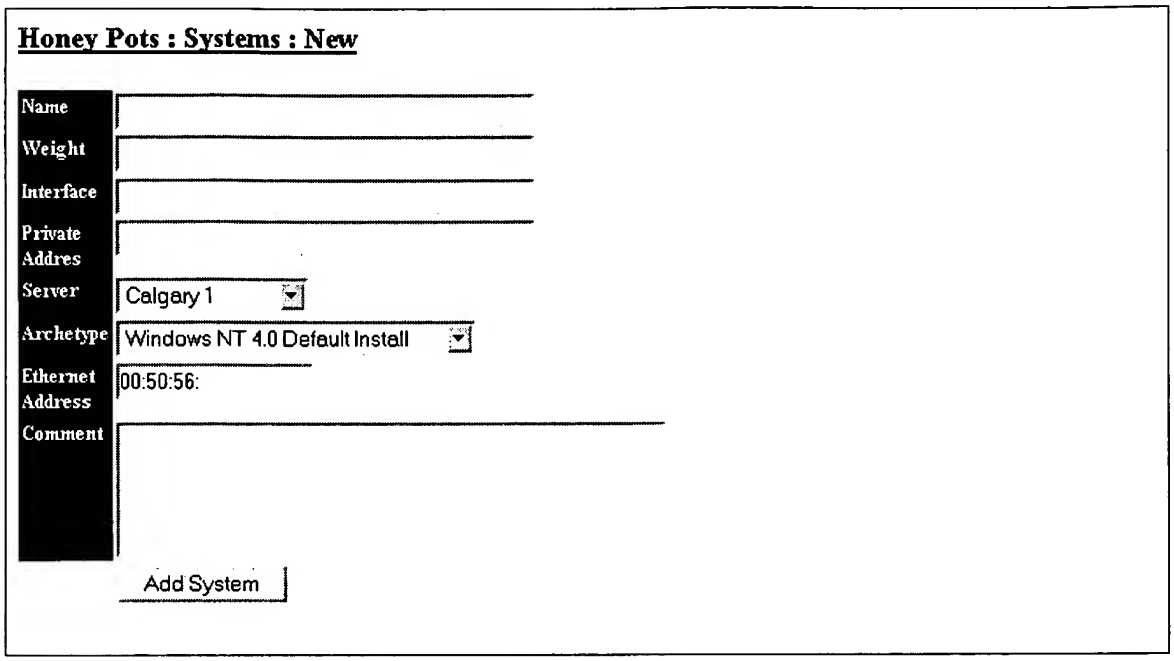
Honey Pots : Systems

[New System] [Show Archived]

Name	State	Server	Weight	Interface	Private Address	Archetype	Ethernet Address	
Windows NT 4.0	Enabled	Calgary 1	10	vmware0	198.169.0.2	Windows NT 4.0 Default Install	00:50:56:00:00:01	[Instances] [Events] [Edit]
Windows 2000	Disabled	Calgary 1	25	vmware1	198.169.1.2	Windows 2000 Default Install	00:50:56:00:00:02	[Instances] [Events] [Edit]
Windows 2000 SP3	Enabled	Calgary 2	35	vmware2	198.169.2.2	Windows 2000 SP3 Default Install	00:50:56:00:00:03	[Instances] [Events] [Edit]
RedHat 7.0	Enabled	Santa Monica 1	30	vmware3	198.169.3.2	RedHat 7.0 Default Install	00:50:56:00:00:04	[Instances] [Events] [Edit]

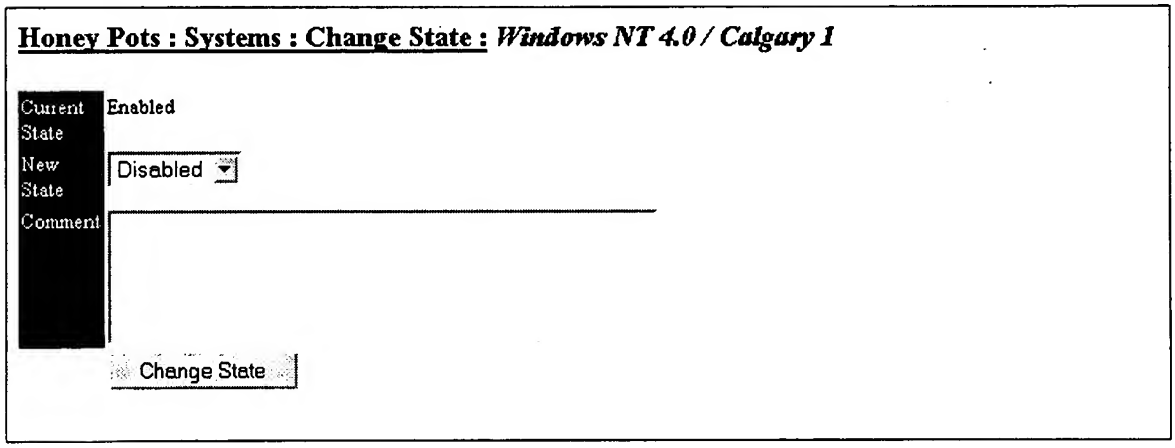
5.4.2.23 Honey Pots : Systems : New

This page permits the administrator create a new honey pot system object.



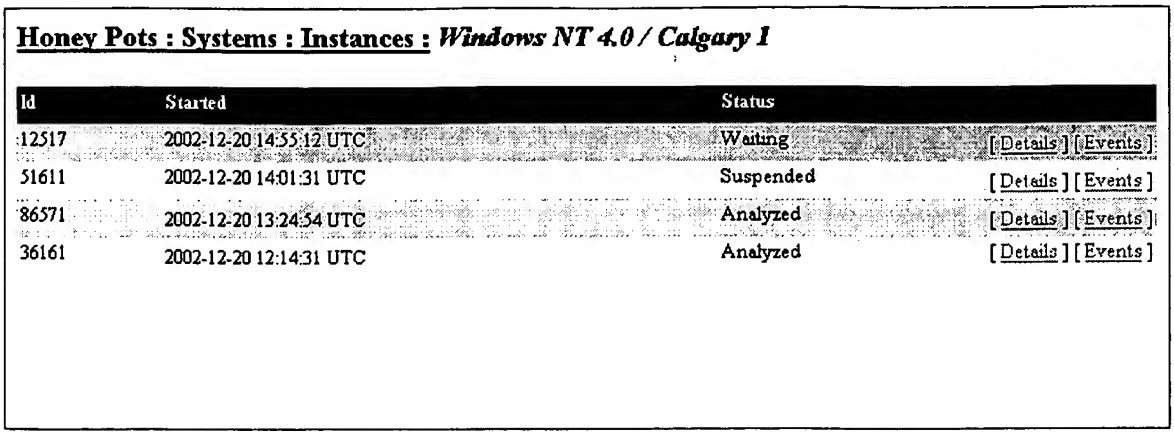
5.4.2.24 Honey Pots : Systems : Change State

This page permits the administrator change the state of a honey pot system object.



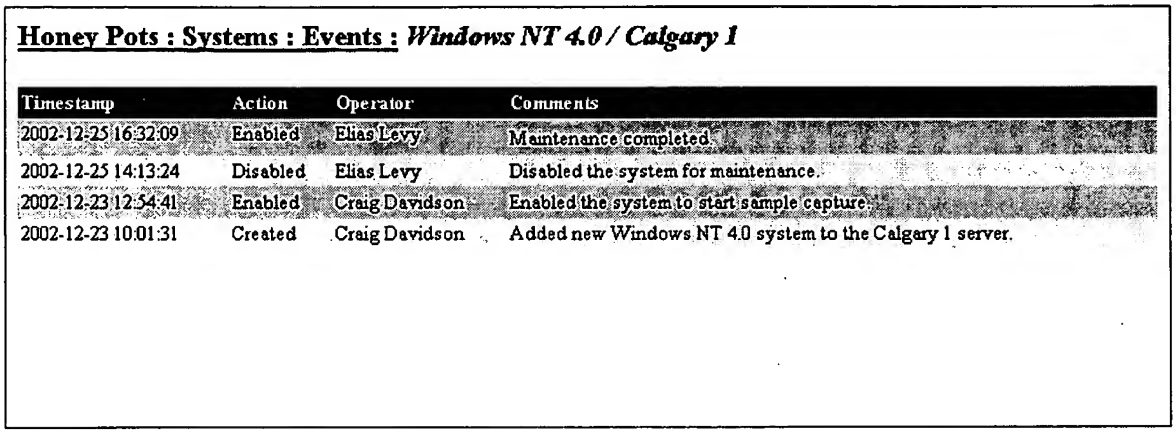
5.4.2.25 Honey Pots : Systems : Instances : <System>

This page permits the administrator view the honey pot instances associated with a honey pot system.



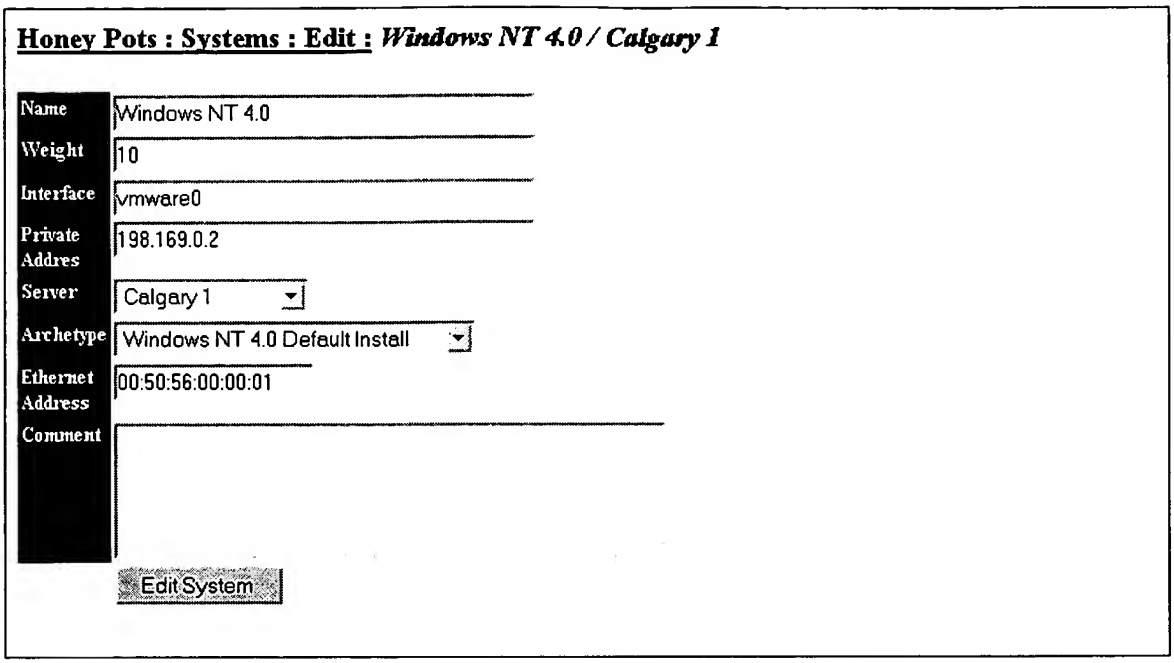
5.4.2.26 Honey Pots : Systems : Events : <System>

This page permits the administrator view the configuration events associated with a honey pot system.



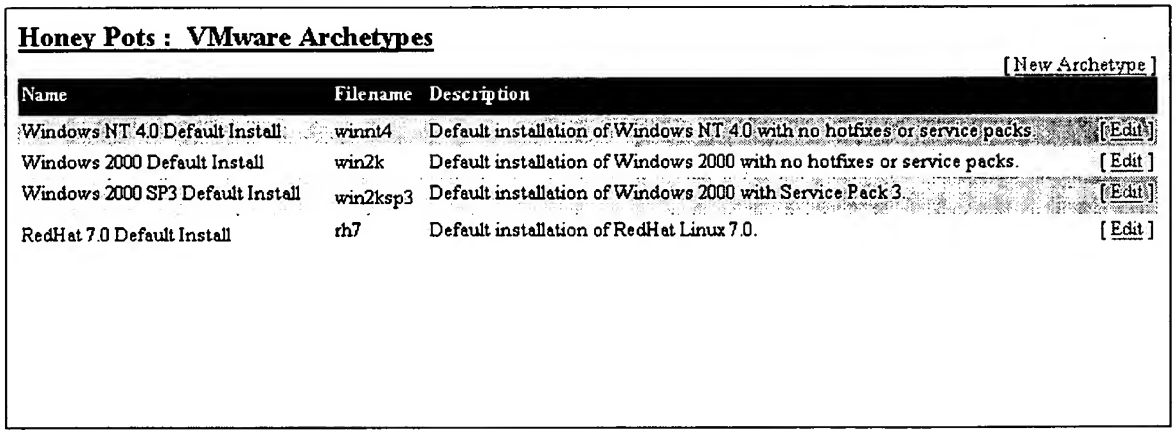
5.4.2.27 Honey Pots : Systems : Edit : <System>

This page permits the administrator edit a honey pot system object.



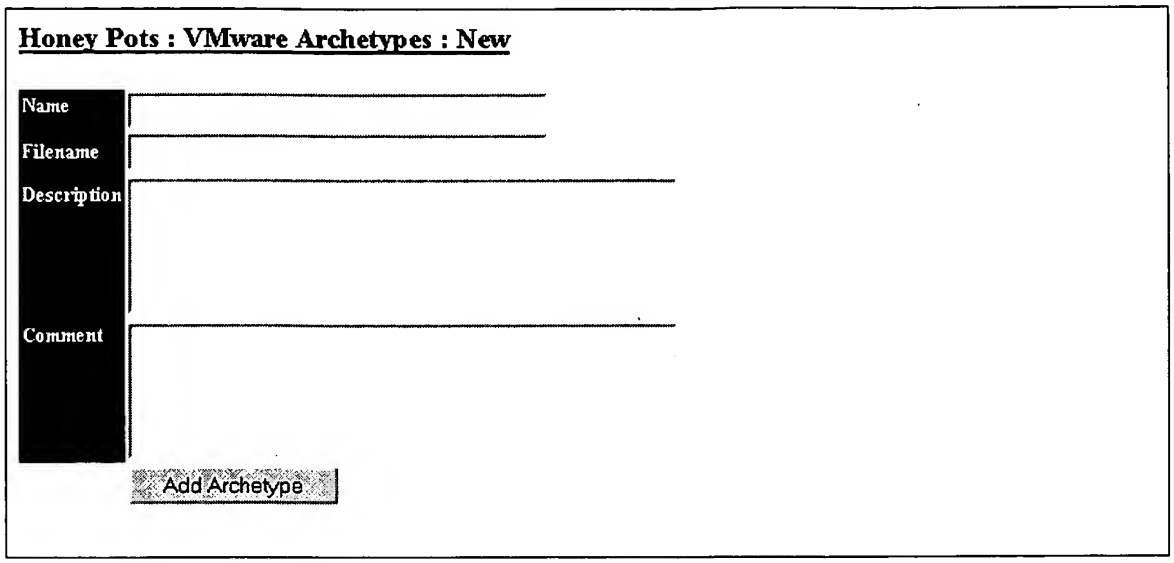
5.4.2.28 Honey Pots : VMware Archetypes

This page permits the administrator view all VMware archetype objects, create new ones, or edit existing ones.



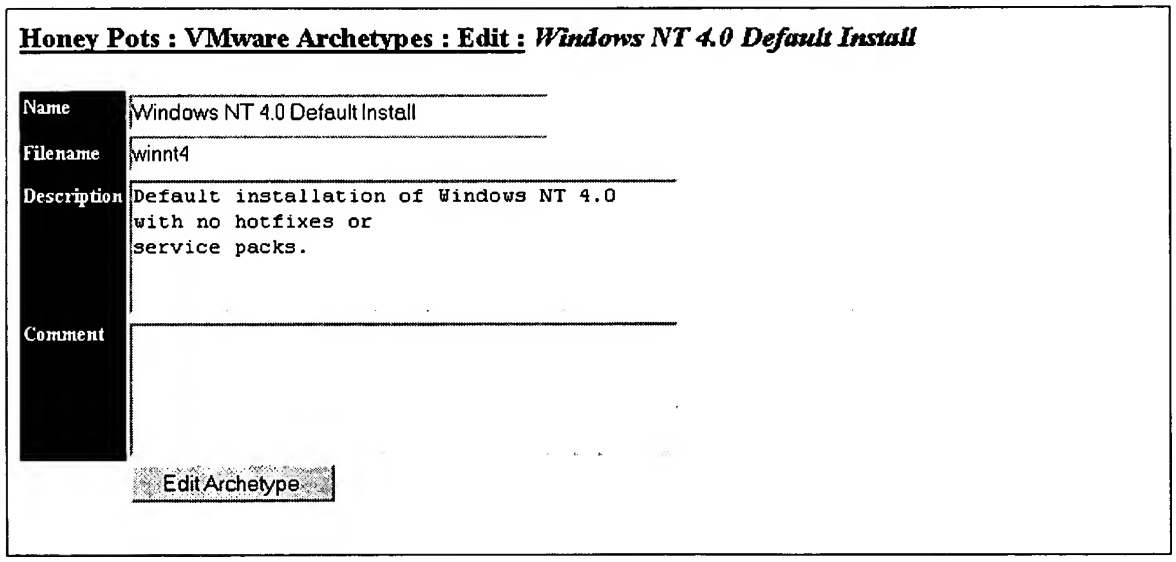
5.4.2.29 Honey Pots : VMware Archetypes : New

This page permits the administrator create a new VMware archetype object.



5.4.2.30 Honey Pots : VMware Archetypes : Edit : <Archetypes>

This page permits the administrator edit a VMware archetype object.



5.4.2.31 Honey Pots : Addresses

This page permits the user view the state of all honey pot IPv4 addresses and what systems they are assigned or bound to.

Honey Pots : Addresses

Address	State	System
198.127.4.0	Unassigned (Available)	
24.131.2.32	Assigned	WinNT 4.0 Honey Pot
24.131.2.33	Bound	RedHat 7.0 Honey Pot
156.24.174.32	Unassigned (Cooling Off)	

5.4.2.32 Honey Pots : Instances

This page permits the user view all honey pot instances, the a honey pot instance's details, or view the events associated with one.

Honey Pots : Instances

[Instance Events]

Id	Started	Status	System	Server	Reviewer	
51251	2002-12-20 14:55:12 UTC	Waiting	Windows NT 4.0 Default Install	Calgary 1		[Details] [Events]
51251	2002-12-20 14:01:31 UTC	Suspended	Windows 2000 SP3 Default Install	Calgary 1		[Details] [Events]
51251	2002-12-20 13:24:54 UTC	Breached	RedHat 7.0 Default Install	Santa Monica 1		[Details] [Events]
51251	2002-12-20 12:14:31 UTC	Analyzed	Windows 2000 Default Install	Calgary 2	Jason Miller	[Details] [Events]
51251	2002-12-20 12:14:14 UTC	Starting	Windows 2000 SP3 Default Install	Santa Monica 1		[Details] [Events]

5.4.2.33 Honey Pots : Instances : Events

This page permits the user view all honey pot instance events.

Honey Pots : Instances : Events

Timestamp	Event	Instance
2002-12-20 13:24:54 UTC	Starting	12312
2002-12-20 13:24:56 UTC	Waiting	63234
2002-12-20 13:42:22 UTC	Contacted	61235
2002-12-20 13:42:41 UTC	Breached	61654
2002-12-20 13:43:12 UTC	Suspended	77163
2002-12-20 13:44:21 UTC	Analyzed	16716

5.4.2.34 Honey Pots : Instances : <Instance>

This page permits the user view a honey pot instance's detailed information, download the instance's data, and post a comment about the instance.

Honey Pots : Instances : *61856*

[Download Instance] [Post Comment] [Close Review]

Started	2002-12-20 13:24:54 UTC
Status	Analyzed

Remote Systems

Address	Hostname (Reverse Lookup)	Operating System (Passive)	Operating System (Active)	Port Numbers
175.15.56.61	adsl-175-15-56-61.pacbell.com	Windows 2000 (4)	Windows 2000 server SP2	TCP: 22, 25, 80 UDP: 53, 6531 (RCP 1003)
206.167.54.76	ftp.wafun.cn	Linux 2.0.34-38	Linux 2.0.34-38	TCP: 21 UDP:

Network Events

Timestamp	Event	Src Address	Dst Address	Src Port	Dst Port
2002-12-20 13:42:23 UTC	BACKDOOR CODERED II root.exe backdoor access attempt	175.15.56.61	206.156.61.6	3461	80
2002-12-20 13:42:25 UTC	BACKDOOR CODERED II root.exe backdoor access	175.15.56.61	206.156.61.6	3462	80
2002-12-20 13:42:26 UTC	297: WEB MISC - http-directory-traversal	175.15.56.61	206.156.61.6	3463	80
2002-12-20 13:42:29 UTC	297: WEB MISC - http-directory-traversal	175.15.56.61	206.156.61.6	3464	80
2002-12-20 13:42:34 UTC	297: WEB MISC - http-directory-traversal	175.15.56.61	206.156.61.6	3465	80
2002-12-20 13:42:41 UTC	spp_http_decode: IIS Unicode attack detected	175.15.56.61	206.156.61.6	3466	80

File System Changes

File	Change	Checksum	MIME Type	Malware	DIS Submit to
C:\WINNT\system.ini	Content changed	MD5: d41d8cd98f00b204e9800998ecf8427e SHA1: A9993E36470681 6ABA3E25717850C26C9CD0D89D	(plain/text)	CA: F-Secure: Kaspersky: McAfee: Norman: TrendMicro:	[Oracle] [DIS]
C:\WINNT\mmc.exe	New file	MD5: 0cc175b9c0f1b6a831c399e269772661 SHA1: 84983E441C3BD26EBAE4AA1F95129E5E54670F1	MS-DOS executable (EXE), OS/2 or MS Windows	CA:Win32.Nimda.A F-Secure:W32/Nimda.A@mm Kaspersky:I-Worm.Nimda McAfee:W32/Nimda.gen@MM Norman:Win32/Nimda.A@mm TrendMicro:PE_NIMDA.A	[Oracle] [DIS]
C:\WINNT\SYSTEM32\riched20.dll	Content changed	MD5: 900150983cd24fb0d6963f7d28e17f72 SHA1: 94AA973CD4C4DAA4F61EEB2BDBAD27218534016F	MS-DOS executable (EXE), OS/2 or MS Windows	CA:Win32.Nimda.A F-Secure:W32/Nimda.A@mm Kaspersky:I-Worm.Nimda McAfee:W32/Nimda.gen@MM Norman:Win32/Nimda.A@mm TrendMicro:PE_NIMDA.A	[Oracle] [DIS]
C:\WINNT\load.exe	New file	MD5: f95b59747cb7938d525a2f31aaf161d0 SHA1: DEA356A2CDDD90C7A7ECEDC5EBB563934F460452	MS-DOS executable (EXE), OS/2 or MS Windows	CA:Win32.Nimda.A F-Secure:W32/Nimda.A@mm Kaspersky:I-Worm.Nimda McAfee:W32/Nimda.gen@MM Norman:Win32/Nimda.A@mm TrendMicro:PE_NIMDA.A	[Oracle] [DIS]

Instance Events

Timestamp	Event
2002-12-20 13:24:54 UTC	Starting
2002-12-20 13:24:56 UTC	Waiting
2002-12-20 13:42:22 UTC	Contacted
2002-12-20 13:42:41 UTC	Breached
2002-12-20 13:43:12 UTC	Suspended
2002-12-20 13:44:21 UTC	Analyzed

Comments

Obviously Nimda

Jason Miller 2002-12-20 13:54:21 UTC

Just another Nimda infection. Bah.

FTP connection

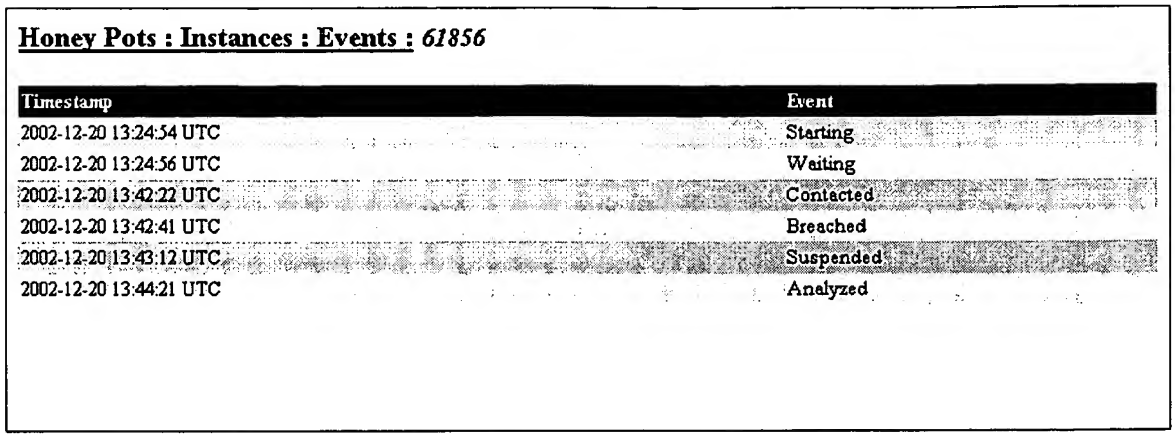
Elias Levy 2002-12-20 14:02:35 UTC

What about that FTP connection? What did it download?

[Post Comment]

5.4.2.35 Honey Pots : Instances : Events : <Instance>

This page permits the user view the events associated with a honey pot instance.

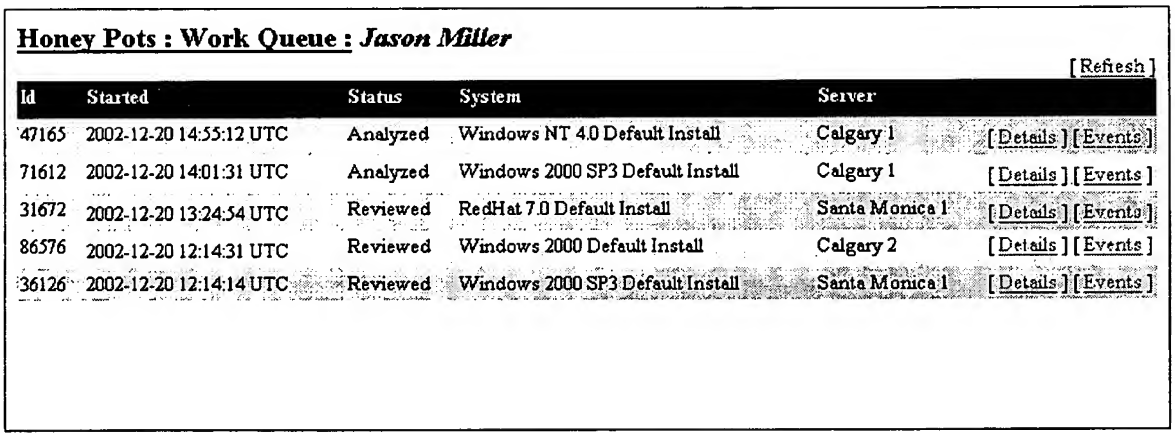


Honey Pots : Instances : Events : ***61856***

Timestamp	Event
2002-12-20 13:24:54 UTC	Starting
2002-12-20 13:24:56 UTC	Waiting
2002-12-20 13:42:22 UTC	Contacted
2002-12-20 13:42:41 UTC	Breached
2002-12-20 13:43:12 UTC	Suspended
2002-12-20 13:44:21 UTC	Analyzed

5.4.2.36 Honey Pots : Work Queue : <Operator>

This page permits the analyst view the honey pot instances that have been assigned to him by the dispatcher for review.



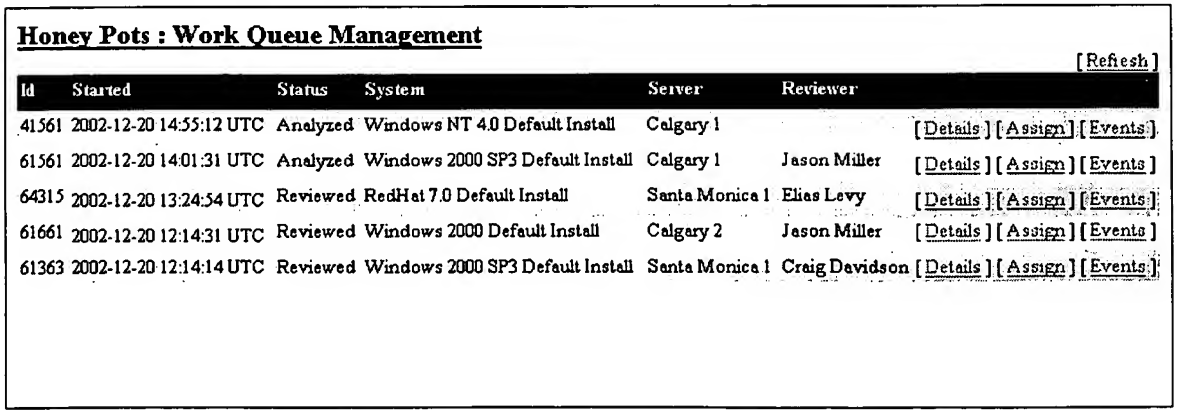
Honey Pots : Work Queue : ***Jason Miller***

[Refiesh]

Id	Started	Status	System	Server	
47165	2002-12-20 14:55:12 UTC	Analyzed	Windows NT 4.0 Default Install	Calgary 1	[Details] [Events]
71612	2002-12-20 14:01:31 UTC	Analyzed	Windows 2000 SP3 Default Install	Calgary 1	[Details] [Events]
31672	2002-12-20 13:24:54 UTC	Reviewed	RedHat 7.0 Default Install	Santa Monica 1	[Details] [Events]
86576	2002-12-20 12:14:31 UTC	Reviewed	Windows 2000 Default Install	Calgary 2	[Details] [Events]
36126	2002-12-20 12:14:14 UTC	Reviewed	Windows 2000 SP3 Default Install	Santa Monica 1	[Details] [Events]

5.4.2.37 Honey Pots : Work Queue Management

This page permits the dispatcher view the list of honey pot instances, who they been assigned for review, and to assign an analyst for review.



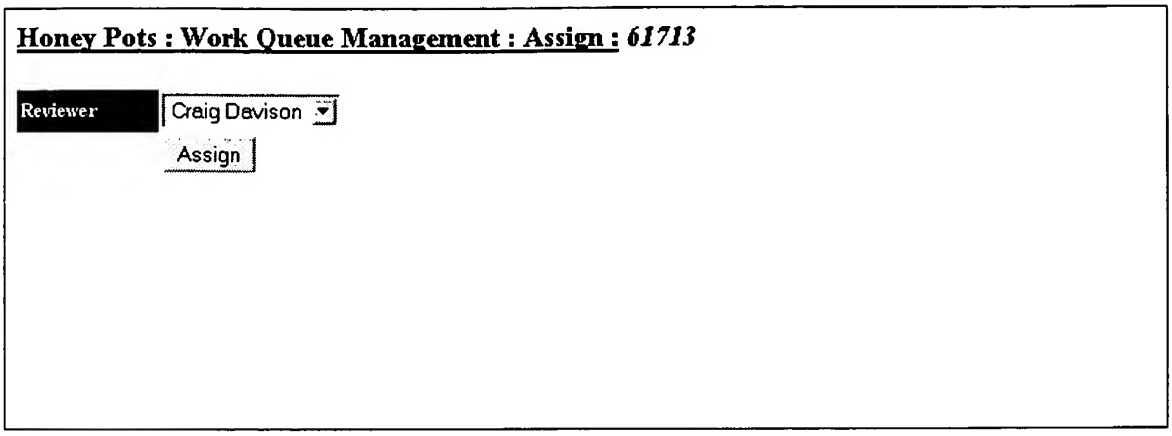
Honey Pots : Work Queue Management

[Refiesh]

Id	Started	Status	System	Server	Reviewer	
41561	2002-12-20 14:55:12 UTC	Analyzed	Windows NT 4.0 Default Install	Calgary 1		[Details] [Assign] [Events]
61561	2002-12-20 14:01:31 UTC	Analyzed	Windows 2000 SP3 Default Install	Calgary 1	Jason Miller	[Details] [Assign] [Events]
64315	2002-12-20 13:24:54 UTC	Reviewed	RedHat 7.0 Default Install	Santa Monica 1	Elias Levy	[Details] [Assign] [Events]
61661	2002-12-20 12:14:31 UTC	Reviewed	Windows 2000 Default Install	Calgary 2	Jason Miller	[Details] [Assign] [Events]
61363	2002-12-20 12:14:14 UTC	Reviewed	Windows 2000 SP3 Default Install	Santa Monica 1	Craig Davidson	[Details] [Assign] [Events]

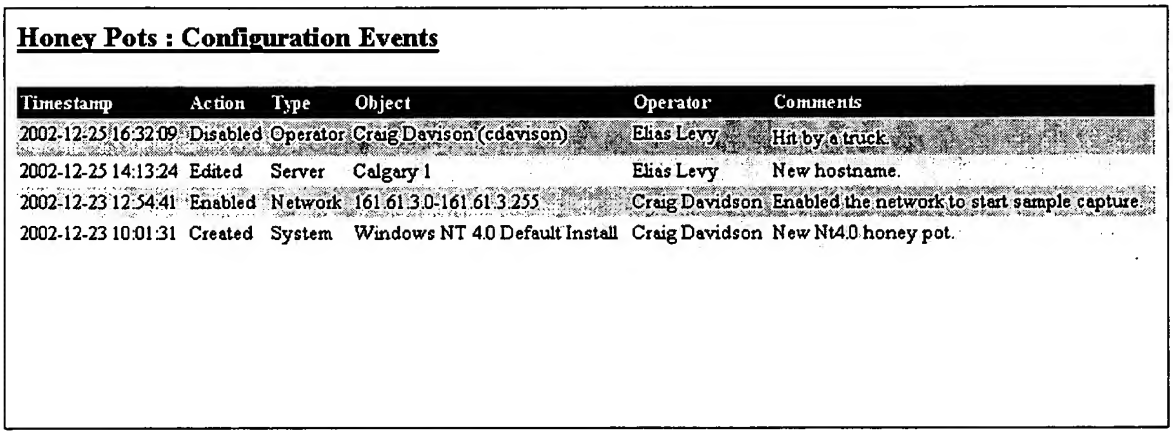
5.4.2.38 Honey Pots : Work Queue Management : Assign : <Instance>

This page permits the dispatcher assign a honey pot instance to an analyst for review.



5.4.2.39 Honey Pots : Configuration Events

This page permits the administrator view all configuration events.



5.5 Database Server

The database server runs MS-SQL for data storage.

IPv4 addresses are stored within the database as integers. The database library layer takes care of converting the binary number in the client's system from its native byte order to that of the database server, and vice versa. Thus, the is not required to perform any explicit byte order operations on the database before storing it or accessing it to interoperate with systems of a different byte order.

5.5.1 Tables

The database maintains the following tables:

5.5.1.1 ObjectStates Table

The **ObjectStates** table maintains a list of constants that describe the different states some system objects can take. They permit an operator to add objects to the system without having the system act on them, and stop the system from acting on some objects without deleting them.

When an object is in the *Enabled* state, the UI displays it normally and the system makes use of it. When an object is in the *Disabled* state, the UI displays it normally, but the system does not make use of it. When an object is in the *Archived* state, the UI onlly displays it if the user explicitly wants to view archived object, and the system does not make use of it. The UI permits a user to purge archived objects, which irrevocably deletes from the database the appropriate records.

ObjectStates			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Object state name: "Enabled", "Disabled", or "Archived".

5.5.1.2 OperatorActions Table

The **OperatorActions** table maintains a list of constants that describe the different actions an operator can perform on the system objects. These include creating an object, enabling it, disabling it, archiving it, editing it, and deleting it.

OperatorActions			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Action name: "Created", "Enabled", "Disabled", "Archived", "Edited", or "Deleted".

5.5.1.3 HoneyPotOperators Table

The **HoneyPotOperators** table maintains a list of accounts that can view and manage honey pot information.

HoneyPotOperators			
ID	Int	Primary Key. Identity.	Identification number.
Account	Varchar	Unique. Not Null.	The account name.

Password	Varchar	Not Null.	The account password.
State	Int	Foreign Key. Not Null.	Reference to **ObjectStates:ID**.
Admin	Bit	Not Null.	Whether the operator is allowed to add operator accounts, changed passwords, etc.
Dispatcher	Bit	Not Null.	Whether the operator is allowed to assign honey pot instances to operators for review.
Analyst	Bit	Not Null.	Whether the operator is allowed to review honey pot instances, post comments, etc.

5.5.1.4 HoneyPotServers Table

The **HoneyPotServers** table maintains a list of systems that manage honey pots.

HoneyPotServers			
ID	Int	Primary Key. Identity.	Identification Number
State	Int	Foreign Key. Not Null.	Reference to **ObjectStates:ID**.
Name	Varchar	Unique. Not Null.	Name of the honey pot server.
Hostname	Varchar	Unique. Not Null.	Hostname or IP address of the honey pot server.
Type	Int	Foreign Key. Not Null.	Reference to **HoneyPotServerTypes:ID**.

5.5.1.5 HoneyPotServerTypes Table

The **HoneyPotServerTypes** table maintains a list of constants that describe the different types of honey pot servers. Initial versions of AQS support only one type of honey pot server, *VMware*. It is foreseen that future versions of AQS will support other honey pot server types, such as *MapTrap*.

HoneyPotServerTypes			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Honey pot server type name: "VMware".

5.5.1.6 HoneyPotNetworks Table

The **HoneyPotNetworks** table maintains a list of routable IPv4 addresses ranges that are mapped to the honey pot systems. IPv4 address ranges must not overlap.

HoneyPotNetworks			
ID	Int	Primary Key. Identity.	Identification number.
State	Int	Foreign Key. Not Null.	Reference to **ObjectStates:ID**.
FirstAddr	Bigint	Unique. Not Null.	First IPv4 address in the IPv4 address range (inclusive). This IPv4 address must be lower within the IPv4 address space than the IPv4 address in the Last column.
LastAddr	Bigint	Unique. No Null.	Last IPv4 address in the IPv4 address range (inclusive). This IPv4 address must be higher within the IPv4 address space than the IPv4 address in the First column.

5.5.1.7 HoneyPotAddresses Table

The **HoneyPotAddresses** table maintains the state of routable IPv4 address available for mapping by honey pot systems.

HoneyPotAddresses			
ID	Int	Primary Key. Identity.	Identification number
Address	Bigint	Unique. Not Null.	A honey pot routable IPv4 address.
State	Int	Foreign Key. Not Null.	Reference to **HoneyPotAddressStates:ID**. The state of the address.
System	Int	Foreign Key.	Reference to **HoneyPotSystems:ID** or *NULL*. The honey pot system the address is mapped to, if any.
Available	Datetime	Not Null. Default Dateadd(second,(-1),Getdate())	The date and time after which an *Unassigned* address, becomes available for assignment. This field is set to a future time to permit *Bound* addresses to "cool off" after they become *Unassigned*.

5.5.1.8 HoneyPotAddressStates Table

The **HoneyPotAddressStates** table maintains a list of constants that describe the different states of routable addresses available for mapping by honey pot systems.

HoneyPotAddressStates			
ID	Int	Primary Key. Identity.	Identification number
Name	Varchar	Unique. Not Null.	Honey pot address state name: "Unassigned", "Assigned", or "Bound".

5.5.1.9 HoneyPotAddressEvents Table

The **HoneyPotAddressEvents** table maintains an audit trail of the honey pot addresses.

HoneyPotAddressEvents			
ID	Int	Primary Key. Identity.	Identification number.
Timestmp	Datetime	Not Null. Default Getdate()	The date and time when the event occurred.
Address	Int	Not Null. Foreign Key.	Reference to **HoneyPotAddresses:ID**.
State	Int	Foreign Key. Not Null.	Reference to **HoneyPotAddressStates:ID**.
Instance	Int	Foreign Key.	Reference to **HoneyPotInstances:ID** if the State is *Bound*, or *NULL*.
System	Int	Foreign Key.	Reference to **HoneyPotSystems:ID** if the State is *Bound*, or *NULL*.

5.5.1.10 HoneyPotSystems Table

The **HoneyPotSystems** table maintains a list of honey pot systems. It maps honey pot systems to honey pot servers.

HoneyPotSystems			
ID	Int	Primary Key. Identity.	Identification number

Server	Int	Foreign Key. Not Null.	Reference to **HoneyPotServers::ID**.
State	Int	Foreign Key. Not Null.	Reference to **ObjectStates:ID**.
Name	Varchar	Unique. Not Null.	Descriptive name of the honey pot system.
Weight	Tinyint	Not Null.	Numeric value that acts as a weight when assigning routable IPv4 addresses to honey pots. All of a honey pot server's honey pot weights are added and the honey pot's weight is used to compute the percentage of routable IPv4 addresses that should be assigned to it.
NetworkInterface	Varchar	Unique. Not Null.	Network interface to the network connected to the honey pot.
PrivateIPv4Address	Bigint	Unique. Not Null.	Private IPv4 address of the honey pot system.

5.5.1.11 HoneyPotInstances Table

The **HoneyPotInstances** table maintains a list of honey pot instances. It maps honey pot instances to honey pot systems.

HoneyPotInstances			
ID	Int	Primary Key. Identity.	Identification number
System	Int	Foreign Key. Not Null.	Reference to **HoneyPotSystems::ID**.
Started	Datetime	Not Null.	Date and time when the instance was created.

Status	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstanceStatus::ID**. Set by the honey pot server. It represents the status the honey pot instance is in.
Reviewer	Int	Foreign Key.	Reference to **HoneyPotOperators::ID** or *NULL*. It represents to operator assigned to review the breach of the honey pot instance.

5.5.1.12 HoneyPotInstanceStatus Table

The **HoneyPotInstanceStatus** table maintains a list of constants that describe the different honey pot instance statuses.

HoneyPotInstanceStatus			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Honey pot instance state name: "Starting", "Executing" "Waiting", "Contacted", "Breached", "Stopping", "Suspended", "Halted", "Analyzed", and "Reviewed".

5.5.1.13 HoneyPotInstanceEvents Table

The **HoneyPotInstanceEvents** table maintains an audit trail of the state of the honey pots.

HoneyPotInstanceEvents			
ID	Int	Primary Key. Identity.	Identification number.
Timestmp	Datetime	Not Null. Default Getdate()	The date and time when the event occurred.
Instance	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstances:ID**.
Status	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstancesStatus:ID**.

Operator	Int	Foreign Key.	Reference to **HoneyPotOperators:ID** or *NULL*. If the Status is *Reviewed*, the operator that reviewed the instance.

5.5.1.14 HoneyPotInstanceComments Table

The **HoneyPotInstanceComments** table maintains a list of comments on honey pot instances by operators.

HoneyPotInstanceComments			
ID	Int	Primary Key. Identity.	Identification number.
Instance	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstances::ID**.
Operator	Int	Foreign Key. Not Null.	Reference to **HoneyPotOperators::ID** or *NULL*. It represents the operator that posted the comment.
Timestmp	Datetime	Not Null.	Date and time when the comment was posted.
Subject	Varchar	Not Null.	The comments subject.
Comment	Varchar	Not Null.	The comment.

5.5.1.15 HoneyPotEvents Table

The **HoneyPotEvents** table maintains an audit trail of actions performed by operators.

HoneyPotEvents			
ID	Int	Primary Key. Identity.	Identification number.
Timestmp	Datetime	Not Null. Default GetDate()	The date and time when the event occurred.
ObjectType	Int	Not Null.	Reference to **HoneyPotObjectTypes:ID**. The object type target of this event.

Object	Int	Not Null.	Reference to **HoneyPotOperators:ID**, **HoneyPotServers:ID**, **HoneyPotNetworks:ID**, or **HoneyPotSystems:ID**.
Operator	Int	Not Null.	Reference to **HoneyPotOperators:ID**. The operator that performed the action that led to the event.
OpAction	Int	Foreign Key. Not Null.	Reference to **OperatorActions:ID**.
Description	Varchar	Not Null.	A description of the action taken generated by the system.
Comment	Varchar		Optional comment entered by the operator describing the changes.

5.5.1.16 HoneyPotObjectTypes Table

The **HoneyPotObjectTypes** table maintains a list of constants that describe the different honey pot object types operators can manage.

HoneyPotObjectTypes			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Honey pot object name: "Operator", "Server", "Network", "System", and "VMwareArchetype".

5.5.1.17 VMwareHoneyPotSystems Table

The **VMwareHoneyPotSystems** table maintains a list of attributes specific to VMware honey pots. It maps a honey pot system to one or more archetypes. The honey pot system must be associated with a honey pot server of type "VMware". The same archetype can be mapped to more than one honey pot system.

VMwareHoneyPotSystems			
ID	Int	Primary Key. Identity.	Identification number.
System	Int	Foreign Key. Not Null. Unique.	Reference to **HoneyPotSystems:ID**.

Archetype	Int	Foreign Key. Not Null.	Reference to **VMwareHoneyPotArchetypes::ID**.
EthernetAddress	Char(17)	Unique. Not Null. Must match the expression '00:50:56:[0-3][0-9A-F]:[0-9A-F][0-9A-F]:[0-9A-F][0-9A-F]'.	Ethernet address of the honey pot system. The address must be of the form *00:50:56:XX:YY:ZZ*, where *XX* is a hex number between 00h and 3Fh, and *YY* and *ZZ* are hex numbers between 00h and FFh.

5.5.1.18 VMwareHoneyPotArchetypes Table

The **VMwareHoneyPotArchetypes** table maintains a list of VMware honey pot archetypes.

VMwareHoneyPotArchetypes			
ID	Int	Primary Key. Identity.	Identification number.
Name	Varchar	Unique. Not Null.	Honey pot archetype name.
Description	Varchar	Not Null.	Description of the archetype, such as what operating system is installed, what applications, their configuration, etc.
Filename	Varchar	Unique. Not Null.	Filename of the archetype.

5.5.1.19 FileSystemChanges Table

The **FileSystemChanges** table maintains a list of changes to the file system of honey pot instances detected during post-intrusion analysis.

A change of *Deleted* indicates the file was removed from the file system. A change of *Metadata* indicates the file's metadata (inode number, permissions, number of links, user-id, group-id, modification time, metadata modification time, ext2 attributes, ACLs, NTFS streams, FAT attributes, etc) were modified. A change of *Content* indicates the file's contents were modified. A change of *Created* indicates the file did not previously exist and was newly created.

FileSystemChanges

ID	Int	Primary Key. Identity.	Identification number.
Instance	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstances:ID**.
Filename	Varchar	Not Null.	Filename of the affected file.
Path	Varchar	Not Null.	Path to the affected file.
Deleted	Bit		Whether the file was deleted.
Metadata	Bit		Whether the file's metadata was modified.
Content	Bit		Whether the file's content was modified.
Created	Bit		Whether the file was created.
Description	Varchar	Not Null.	Description of the changes if the *Metadata* flag is set (e.g. "user-id changed from root to bin").
MD5	Char(32)		The MD5 hash of the file's contents, if the Change flags *Content* or *Created* are set, or *NULL*.
SHA1	Char(40)		The SHA1 hash of the file's contents, if the Change flags *Content* or *Created* are set, or *NULL*.
Type	Varchar		A description of the type of the file, if the Change flags *Content* or *Created* are set, or *NULL*.
MIMEType	Varchar		The MIME type of the file, if the Change flags *Content* or *Created* are set, or *NULL*.
Malware	Varchar		The type of malware the file is a sample of, if the Change flags *Content* or *Created* are set, or *NULL*.
DIS	Varchar.		Result from DIS, if the Change flags *Content* or *Created* are set, or *NULL*

5.5.1.20 RemoteSystems Table

The **RemoteSystems** table maintains a list of remote systems seen communicating with a honey pot instance.

RemoteSystems			
ID	Int	Primary Key. Identity.	Identification number.
Instance	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstances:ID**.
Address	Bigint	Not Null.	IPv4 address of the remote system.
Hostname	Varchar		Hostname associated to the remote system by performing a reverse DNS query on the remote system's IPv4 address, or *NULL* if the query returned no result.
PassiveOS	Varchar		Operating system of the remote system estimated via passive fingerprinting, or *NULL* if undetermined.
ActiveOS	Varchar		Operating system of the remote system estimated via active fingerprinting, or *NULL* if undetermined.
Uptime	Varchar		System uptime, or *NULL* if undetermined.
TCPSequence	Varchar		The difficulty of guessing the system's TCP sequence numbers, or *NULL* if undetermined.

5.5.1.21 RemoteSystemPorts Table

The **RemoteSystemPorts** table maintains a list of accessible ports in a remote system.

RemoteSystemPorts			
RemoteSystem	Int	Foreign Key. Not Null.	Reference to **RemoteSystems:ID**.
Port	Int	Not Null.	The port number.
TCP	Bit	Not Null.	If it is not TCP, then it must be UDP.

State	Int	Foreign Key. Not Null.	Reference to **RemoteSystemPortStates:ID**.
Ident	Varchar		Information returned by ident for the port number, or *NULL* if undetermined.
RPC	Int		If an RPC service is listening in the *Open* port, its RPC program number and version, or *NULL* if the port is *Unfiltered*, *Filtered* or *Closed*, or no RPC service is listening on it.

5.5.1.22 RemoteSystemPortStates Table

The **RemoteSystemPortStates** table maintains a list of constants that describe the different states of a port.

HoneyPotAddressStates			
ID	Int	Primary Key. Identity.	Identification number
Name	Varchar	Unique. Not Null.	Port state name: "Open", "Closed", "Filtered", or "Unfiltered".

5.5.1.23 NetworkEvents Table

The **NetworkEvents** table maintains a list of network events (probes and attacks) detected by an IDS against a honey pot instance.

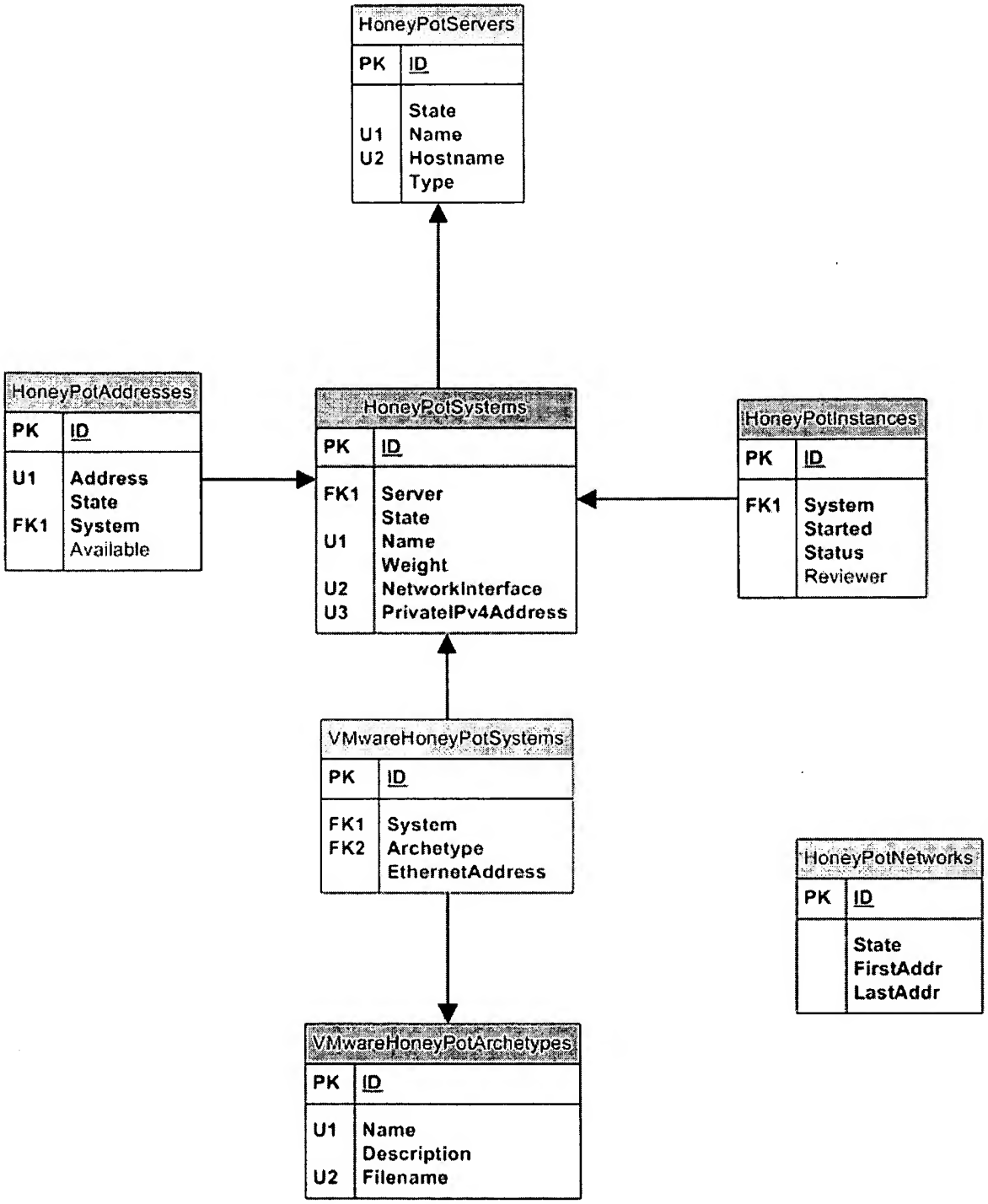
NetworkEvents			
ID	Int	Primary Key. Identity.	Identification number
Instance	Int	Foreign Key. Not Null.	Reference to **HoneyPotInstances:ID**.
Timestmp	Datetime	Not Null.	The date and time of the event.
SnortID	Int	Not Null.	The snort signature id that matched the event.
Revision	Smallint	Not Null.	The snort signature revision that matched the event.

Message	Varchar	Not Null.	The snort message.
SourceAddress	Bigint	Not Null.	The source address of the matched IPv4 traffic.
DestinationAddress	Bigint	Not Null.	The destination address of the matched IPv4 traffic.
Protocol	Varchar		The protocol that generated the event.
SourcePort	Int		The source port number if the event was generated by TCP or UDP traffic.
DestinationPot	Int		The destination port number of the event was generated by TCP or UDP traffic.

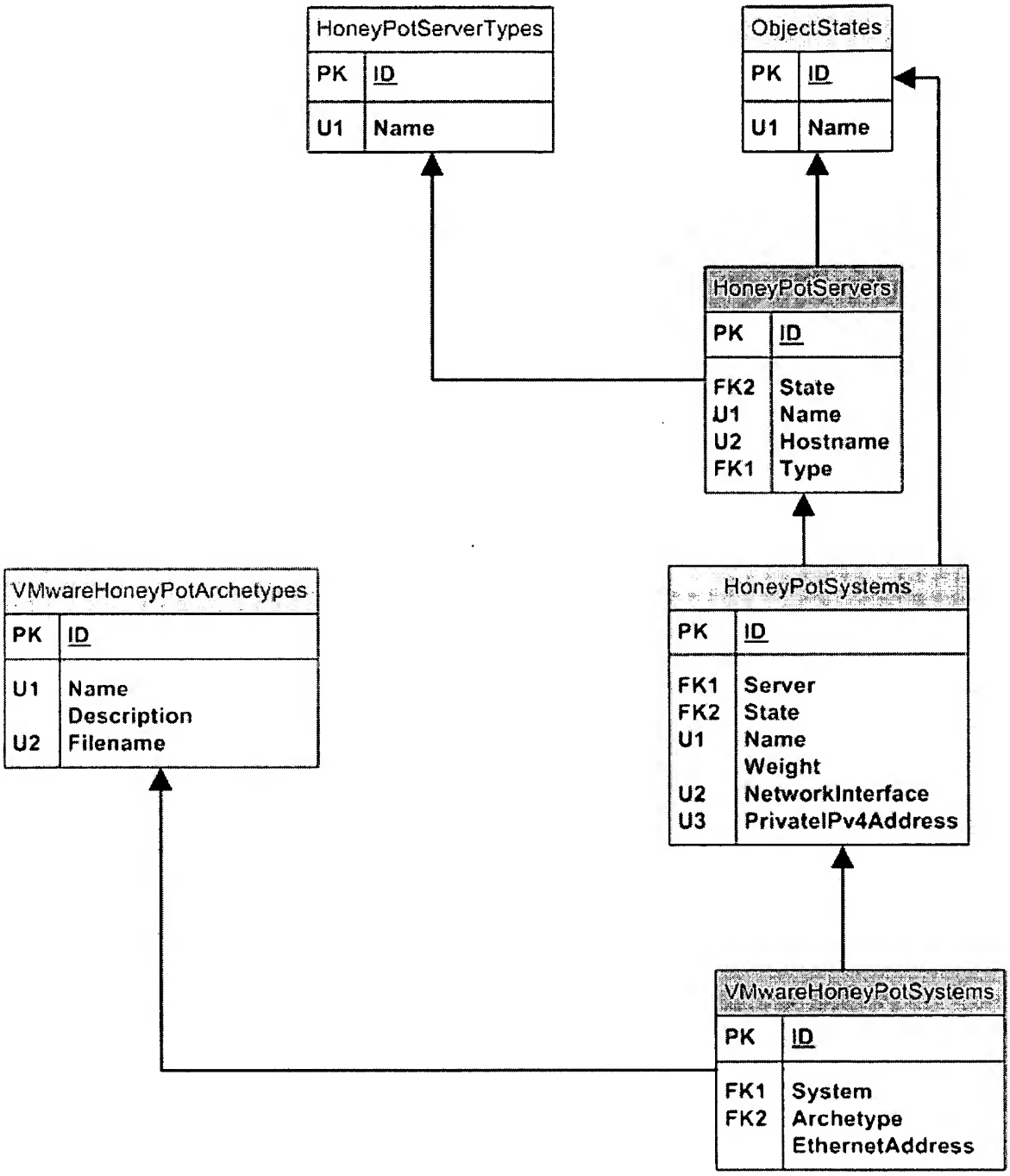
5.5.2 Entity Relationship Diagrams

The following are partial entity relationship diagrams (ERDs) of the tables described earlier.

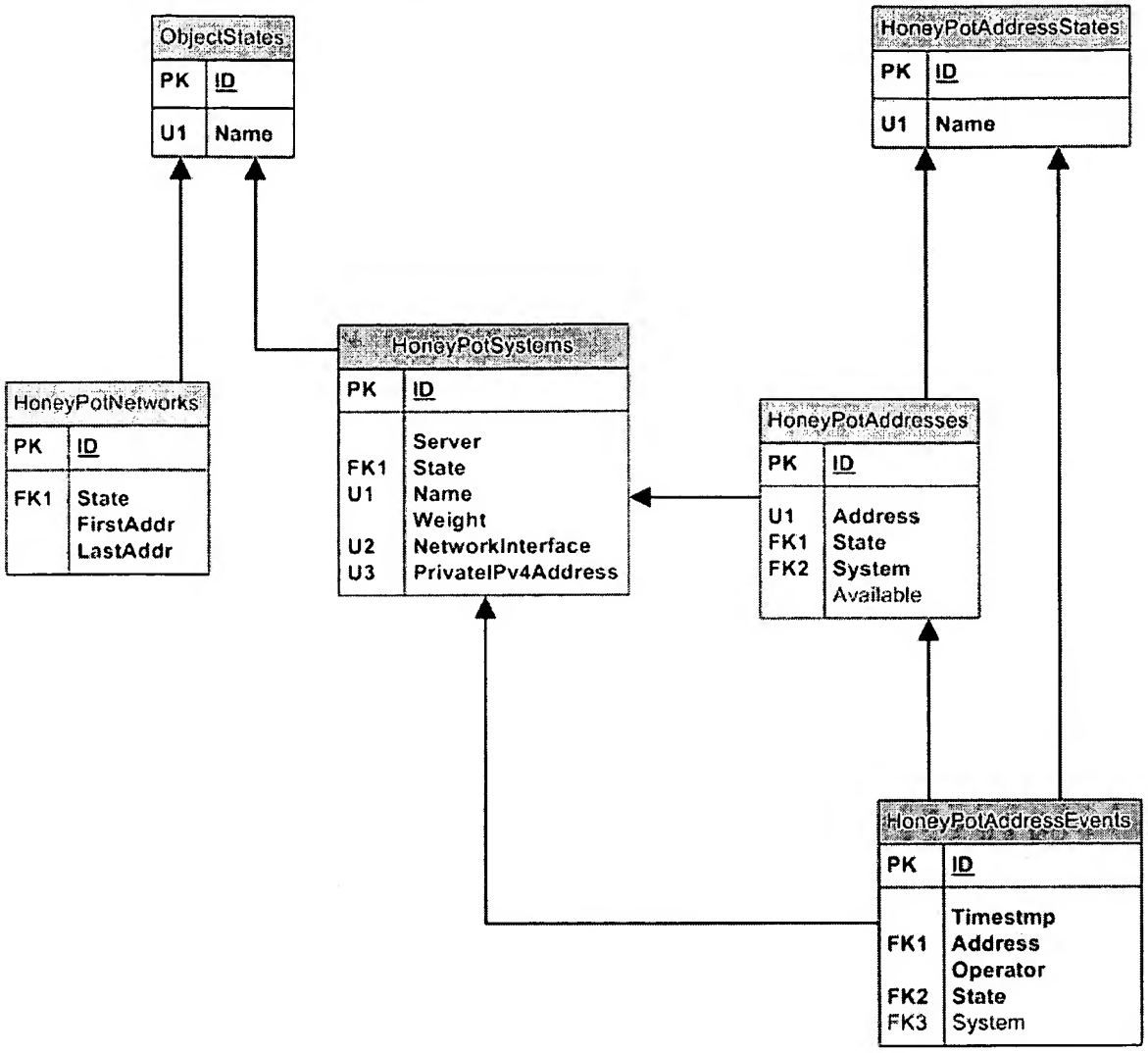
5.5.2.1 Major Tables



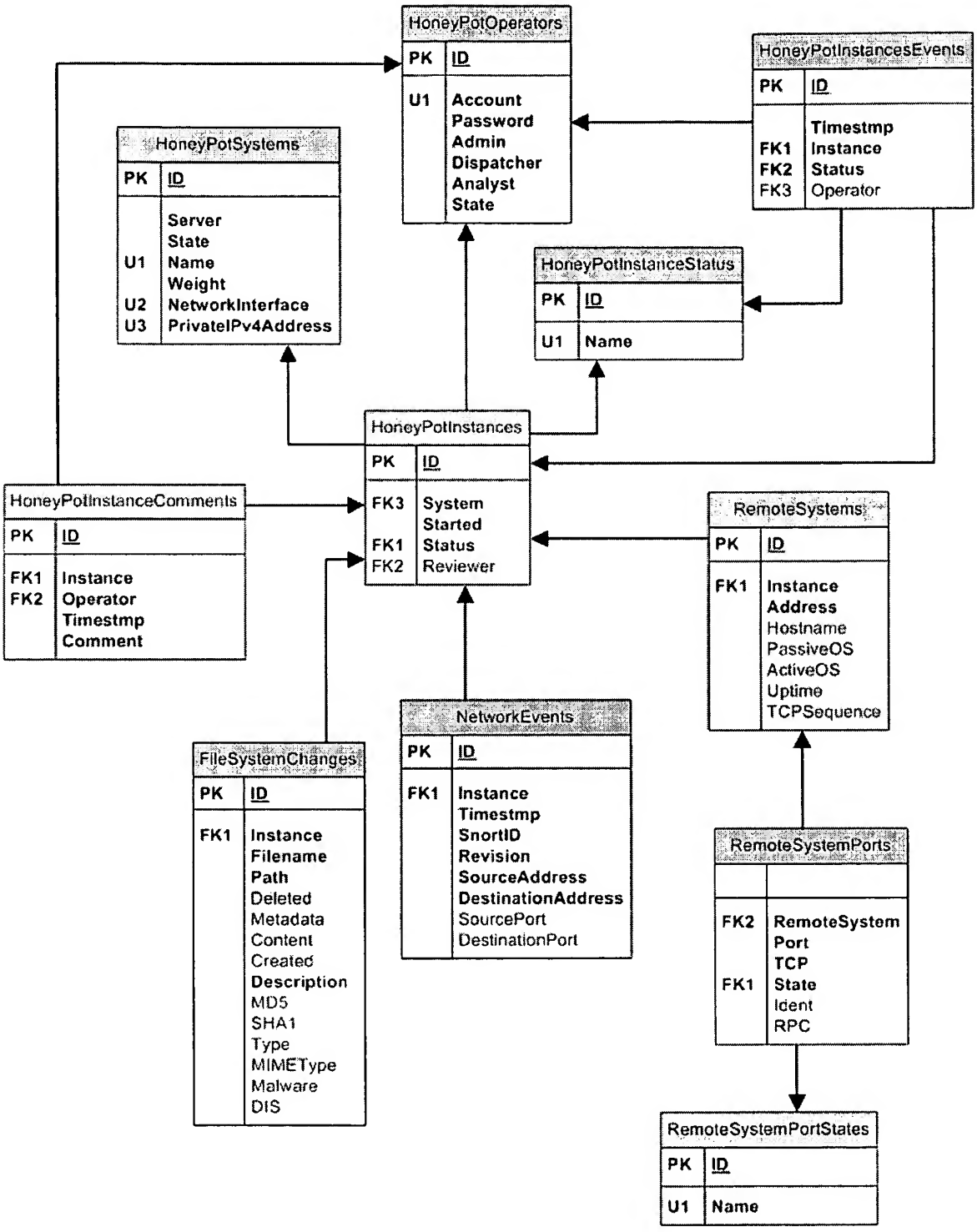
5.5.2.2 Honey Pot Servers and Honey Pot Systems



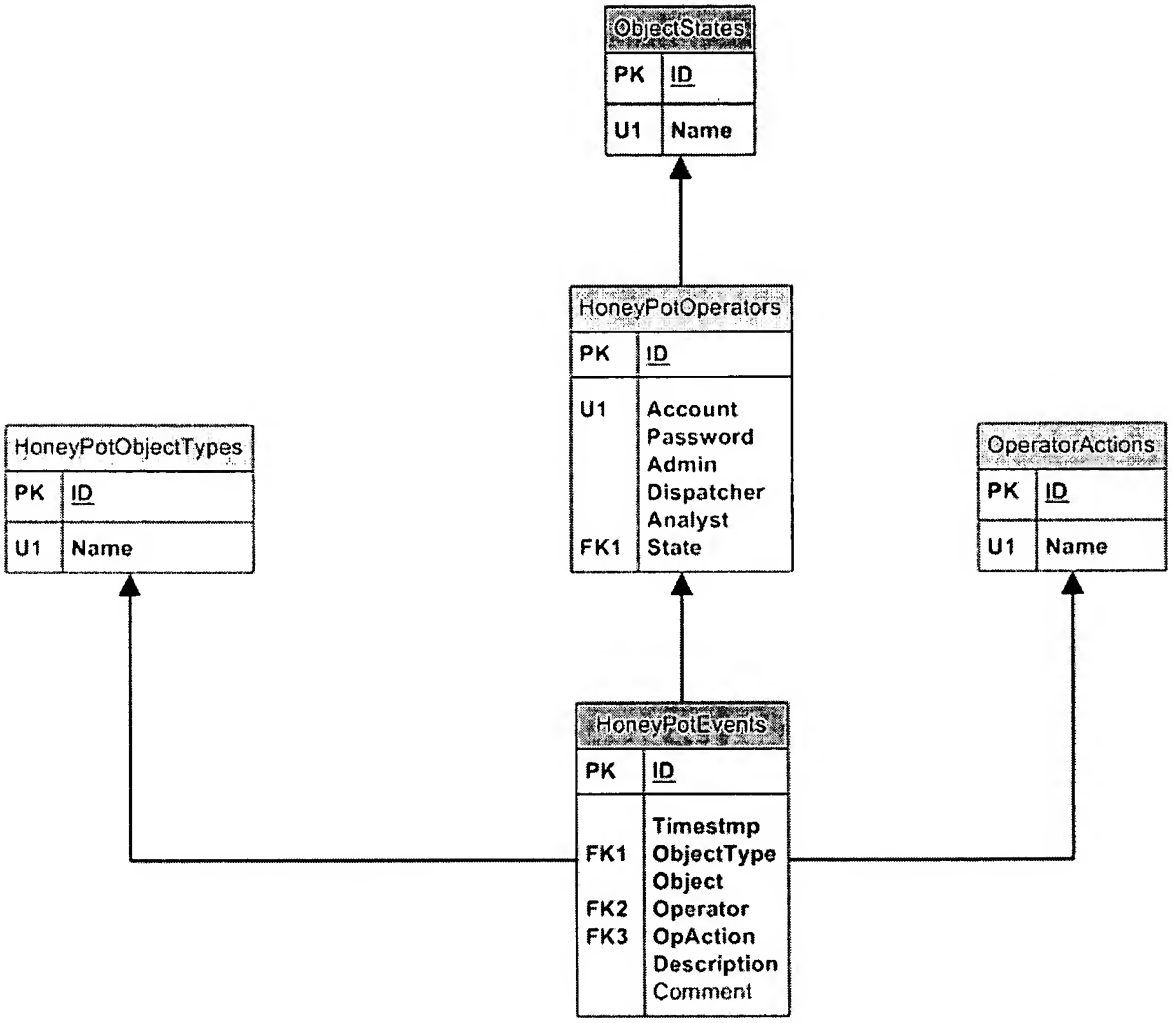
5.5.2.3 Honey Pot Addresses



5.5.2.4 Honey Pot Instances



5.5.2.5 Operators and Events



6. External Dependencies

6.1 DeepSight TMS

The DeepSight TMS team is building a database of TCP and UDP port number information, as well as RPC program number information. Once their work is complete, we will integrate with them in a future revision by creating a link that opens a small window from a remote system port numbers' in an instance's detailed information which displays information associated with the port number such as its associated protocols, description, and known vulnerabilities.

6.2 DIS

If the Santa Monica team wishes to receive not only captures executable samples, but also any ancillary data (e.g. packet dumps), then they need to define a file format to capture such data that can be processed by DIS. For example, we could archive the captured executable with the ancillary data in a ZIP format file.

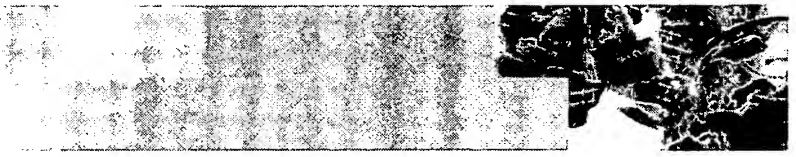
Exhibit B

symantec.

DeepSight Attack Quarantine System (AQS)

Elias Levy
Architect
Symantec Security Response

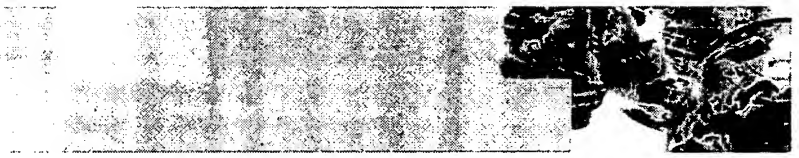
symantec.



Problem

- Analysis of malicious code can't start in earnest until the capture of samples.
- Samples are obtained from:
  - Customers through DIS/SND.
    - ❖ Reactive.
  - Sample sharing groups.
    - ❖ Can be unreliable.
    - ❖ Do not represent a formal relationship.
  - Temporary honey pots
    - ❖ Take time to set up &manage.
    - ❖ Do not have a large network footprint.

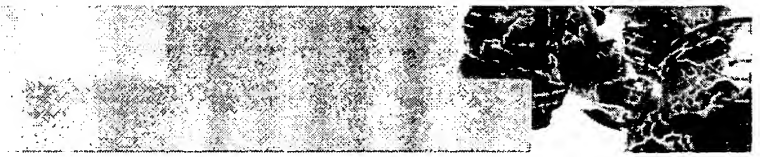
symantec.



Goals

- Early detection and capture of new malicious code in the wild.
- Early detection of new attacks or vulnerabilities in the wild, and capture of their associated tools or exploits.
- Integration into the existing Symantec malicious code capture and analysis infrastructure.
- Automate some of the analysis process.
- 24/7 operation.

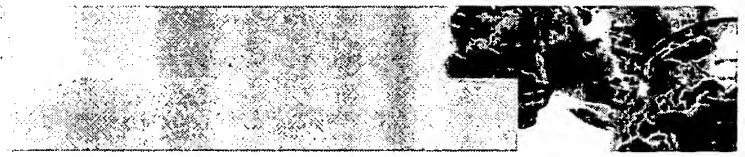
symantec.



Benefits

- Permit earlier analysis, alerting, and signature development.
- Ease our dependence on customers submitting samples and sample sharing groups.

symantec.

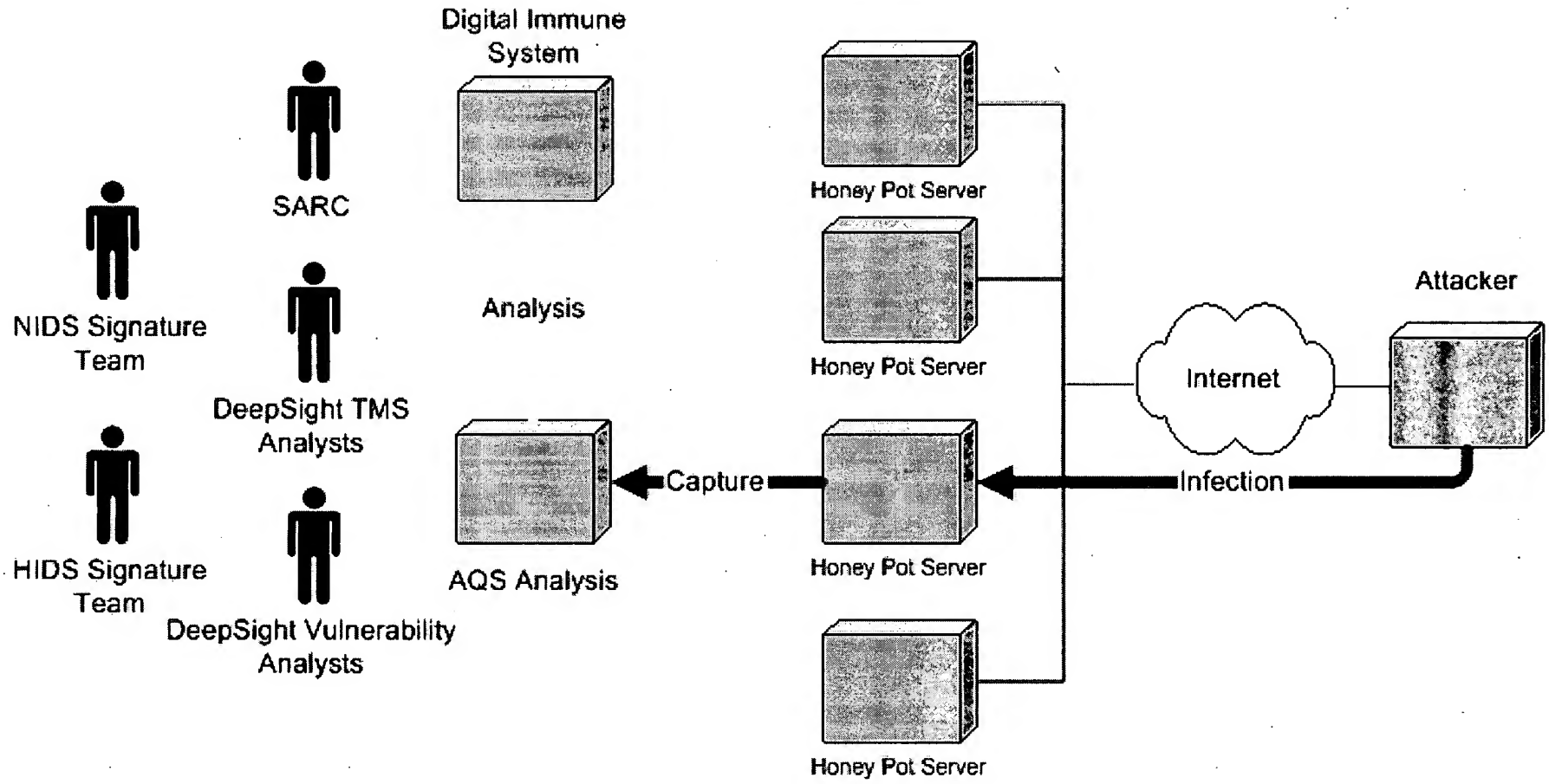


Solution

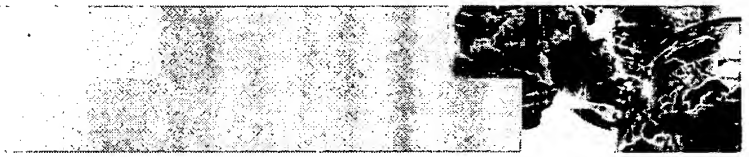
- Network of Automated Honey Pots
  - Large network footprint increases probability of early capture.
  - Automated honey pot deployment and sample capture.
  - Partially automated breach analysis.
  - Feeds sample to existing Symantec Digital Immune System.

symantec.

Diagram



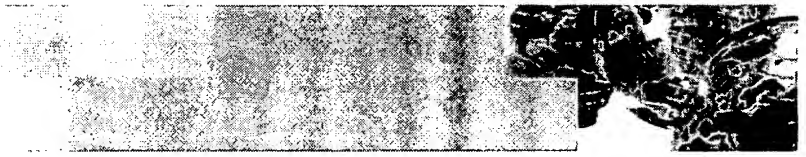
symantec.



Network Address Translation Subsystem

- Performs dynamic M-N mapping of routable IP addresses to private IP addresses.
- Honey pot systems *weights* determines what percentage of available IP addresses are assigned to a honey pot system.

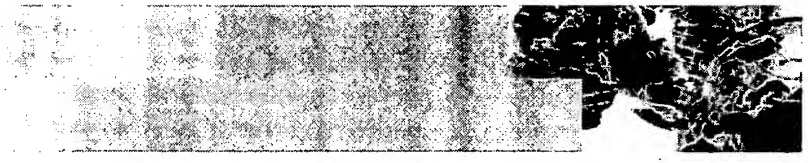
symantec.



Egress Traffic Control Subsystem

- Inline IDS protects external computers against attack by breached honey pots by dropping connections that match known attack signatures.
- Also used to drop some incoming attacks.

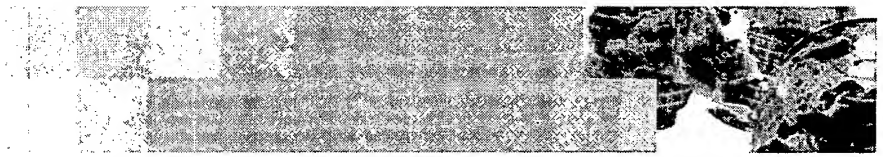
symantec.



Honey Pot Monitoring & Management Subsystem

- Creates new instances of honey pot systems.
- Determines when a honey pot instance has been breached.
  - A honey pot is assumed breached when:
    - An outgoing connection is detected.
    - A timeout occurs.
- Shutdowns breached honey pot instances after:
  - A maximum number of outgoing connections.
  - A timeout.

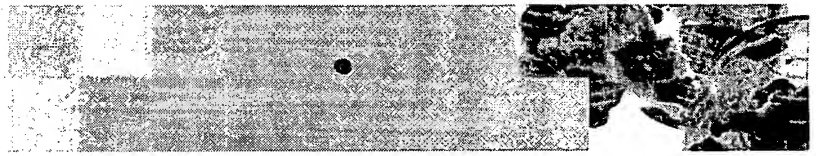
symantec.



Honey Pot Network
Traffic Capture Subsystem

- Records all network traffic from/to a honey pot instance.

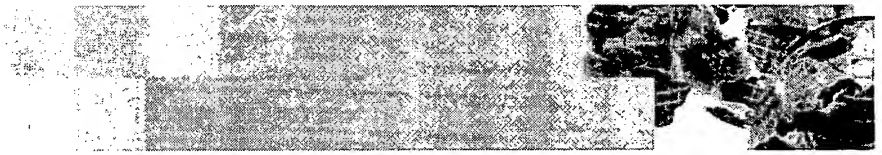
symantec.



Network Access Control Subsystem

- Drops:
  - Spoofed packets from a honey pot instance.
  - All cross-honey pot network traffic.
  - All network traffic from a honey pot instance to the honey pot server.
  - All network traffic from a honey pot instance to the support network, with some exceptions.
  - Network traffic destined to some high-risk port (e.g. 139, 445).
- Performs rate limiting of outgoing network traffic.

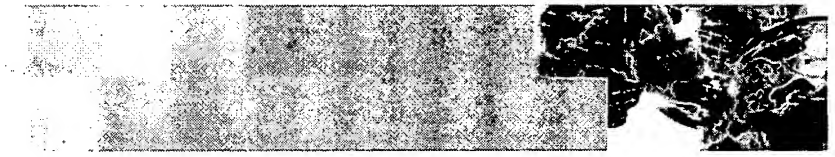
symantec.



Honey Pot Support Services Subsystem

- Provides the honey pot instances with DHCP service.
- Provides the honey pot instances with DNS service.

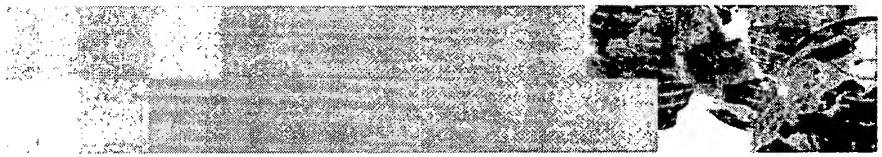
symantec.



Post-Intrusion Automated Analysis Subsystem

- Performs some automated analysis of breached honey pot instances:
  - Detects file system changes.
  - Estimates the file type of new or modified files.
  - Determines whether new or modified files represent or are infected by known malware.
  - Generates a timeline of file system changes.
  - Displays a summary of network traffic flows and extracts TCP flow data.
  - Analyzes the network traffic for known attacks.
  - Estimates the operating system of remote hosts by analyzing the network traffic.
- Submits new or modified executable files to DIS.

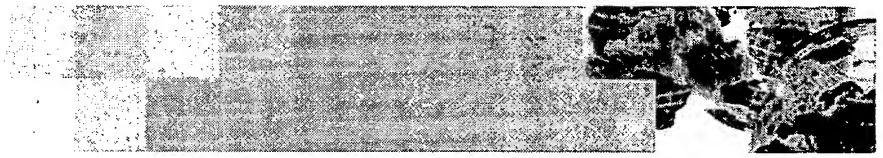
symantec.



Malware Oracle

- Examines submitted files with a number of AV scanners and reports their results.

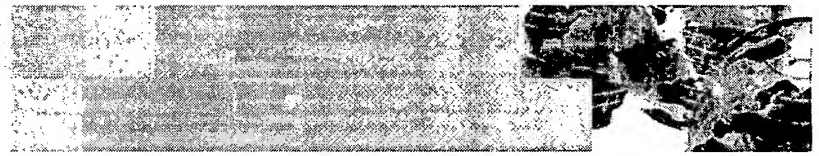
symantec.



Web-based Interface

- Permits operators to:
  - Configure the public address space.
  - Configure honey pot servers.
  - Configure honey pot systems.
- Permits analysts to:
  - Review breached honey pot instances.
  - Download new or modified files.
  - Download captured network traffic.
  - Submit files to the Malware Oracle.
  - Submit files to DIS.

symantec.



Account: aleph1 | **Logout**

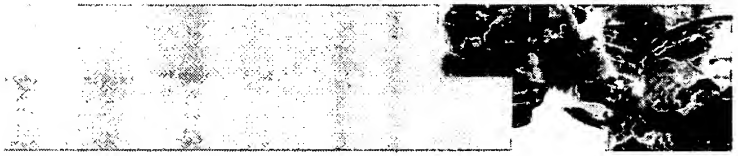
Honey Pots : Work Queue Management

[Instance Events] [Work Queue] [Refresh]

There is enough free space left for **at least 7** new Analyzed Instances.

ID	Classification	Started (GMT)	Status	System	Server	Reviewer	Archive		
607	New	2003-07-28 11:00:01	Analyzed	36: Win2k professional #1	matrix01	**bartek**		[Details]	[2 Comments]
1342		2003-09-17 19:34:29	Analyzed	37: Win2k Adv Server #1	matrix01	[Assign]	Delete	[Details]	[Comments]
1331		2003-09-17 01:05:50	Analyzed	37: Win2k Adv Server #1	matrix01	[Assign]	Delete	[Details]	[Comments]
1330		2003-09-16 23:50:15	Analyzed	38: Win2k Adv Server + MSSQL #1	matrix01	[Assign]	Delete	[Details]	[1 Comment]
1328		2003-09-16 22:59:48	Analyzed	38: Win2k Adv Server + MSSQL #1	matrix01	[Assign]	Delete	[Details]	[1 Comment]
1326		2003-09-16 22:24:01	Analyzed	38: Win2k Adv Server + MSSQL #1	matrix01	[Assign]	Delete	[Details]	[Comments]
1325		2003-09-16 21:53:15	Analyzed	38: Win2k Adv Server + MSSQL #1	matrix01	[Assign]		[Details]	[1 Comment]
1317		2003-09-16 15:38:02	Analyzed	38: Win2k Adv	matrix01	[Assign]		[Details]	[Comments]

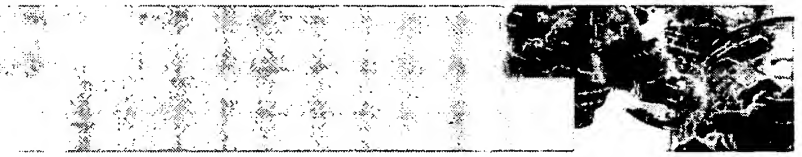
symantec.



Filesystem Changes

File	Change	Type	Malware	DIS	Tools
/WINNT	Metadata				
/WINNT/hfn1.tmp	New File	MS-DOS executable (EXE), OS/2 or MS Windows	F-Prot for Linux: W32/Parite.A (exact) Sophos Anti-Virus for Unix: W32/Parite-A AntiVirus Command Line Scanner: W32.Pinfi F-Prot for DOS: W32/Parite.A (exact) AVP for DOS: packed.UPX AVP for DOS: Win32.Parite.a Sophos Anti-Virus for DOS/Windows 3.1: W32/Parite-A	norepair	[Oracle] [DIS]
/WINNT/system32	Metadata				
/WINNT/system32/10.BAT	New File	ASCII text, with CRLF line terminators	Kaspersky Anti-Virus for Linux: Worm.Win32.Muma.a AntiVirus Command Line Scanner: BAT.Mumu.A.Worm AVP for DOS: Worm.Win32.Muma.a		[Oracle] [DIS]
/WINNT/system32/hack.bat	New File	ASCII text, with CRLF line terminators	Kaspersky Anti-Virus for Linux: Worm.Win32.Muma.a AntiVirus Command Line Scanner: BAT.Mumu.A.Worm AVP for DOS: Worm.Win32.Muma.a		[Oracle] [DIS]
/WINNT/system32/ipc.bat	New File	ASCII text, with no line terminators	F-Prot for Linux: BAT/Muma.A (exact) Kaspersky Anti-Virus for Linux: Worm.Win32.Muma AntiVirus Command Line Scanner: BAT.Mumu.A.Worm F-Prot for DOS: BAT/Muma.A (exact) AVP for DOS: Worm.Win32.Muma		[Oracle] [DIS]
/WINNT/system32/ipcNL.exe	New File	MS-DOS executable (EXE), OS/2 or	F-Prot for Linux: W32/Parite.A Kaspersky Anti-Virus for Linux: Win32.Parite.a Sophos Anti-Virus for Unix: W32/Parite-A AntiVirus Command Line Scanner: W32.Pinfi	infected	[Oracle] [DIS]

symantec.



Account: aleph1 | **Logout**

Honey Pots : Instances : *738*

[Download Traffic] [Post Comment]

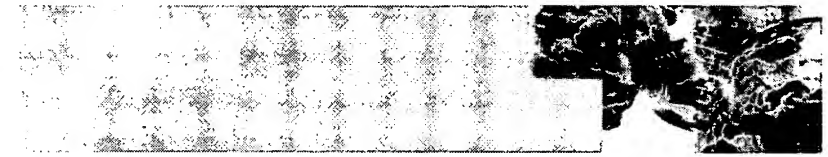
Started (GMT)	2003-08-06 21:12:15
Status	Analyzed
System	35: NT4 workstation #1
Private IP Address	192.168.4.2
Mark Instance	[New] [Known] [**Undefined**]
Reviewer	jroculan

Filesystem Changes	Filesystem Timeline	Remote Systems	Network Flows	Network Events	Instance Events	1 Comment

Filesystem Timeline

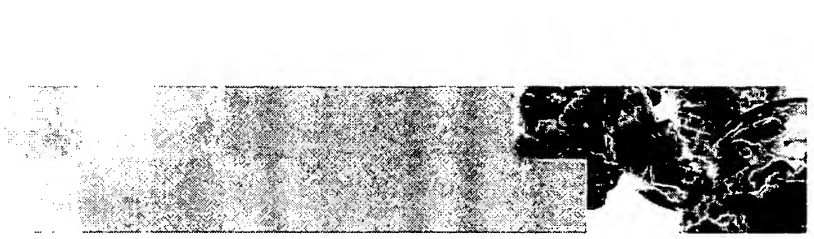
Timestamp	Event	Deleted	New	Filename
2003-08-06 15:44:42	Accessed (a)	N	Y	/WINNT/system32/qjinfo.exe
2003-08-06 15:44:43	Metadata Changed (c)	N	Y	/WINNT/system32/qjinfo.exe
2003-08-06 15:45:10	Accessed (a)	N	N	/WINNT/system32/RPCLTC1.DLL
2003-08-06 15:45:12	Accessed (a)	N	N	/WINNT/system32/WS2_32.DLL
2003-08-06 15:45:12	Modified (mc)	N	N	/WINNT
2003-08-06 15:45:12	Metadata Changed (ac)	N	Y	/WINNT/system32/LAST.EXE
2003-08-06 15:45:12	Accessed (a)	N	N	/WINNT/system32/wsock32.dll
2003-08-06 15:45:12	Accessed (a)	N	N	/WINNT/system32/MPR.DLL

symantec.



Remote Systems

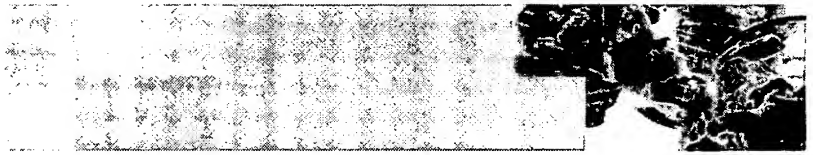
Address	Hostname (Reverse Lookup)	Operating System (Passive)	Operating System (Active)	Port Numbers
4.40.103.64	lsanca1-ar43-4-40-103-064.lsanca1.dsl-verizon.net	Windows 2000 (9)	-	TCP: UDP:
10.21.160.15		35040:32:1460:1:0:1:1:64	-	TCP: UDP:
12.5.86.238		-	-	TCP: UDP:
12.80.24.151	151.los-angeles-61-62rs.ca.dial-access.att.net	Windows 2000, 98, NT 5.0, NT 5.1	-	TCP: UDP:
12.149.96.228	abcnfuse.abctruck.com	Windows NT 5.1	-	TCP: UDP:
12.251.107.193	12-251-107-193.client.attbi.com	55808:107:1460:0:2:1:1:52	-	TCP: UDP:
24.247.222.243	24.247.222.243.kzo.mi.chartermi.net	Windows 98, NT 5.0, NT 5.1	-	TCP: UDP:
32.100.238.153	slip-32-100-238-153.ca.us.prserv.net	Windows 2000, 98, NT 5.0, NT 5.1	-	TCP: UDP:
33.76.249.77		55808:110:1460:0:2:1:1:52	-	TCP: UDP:
59.179.14.163		55808:111:1460:0:2:1:1:52	-	TCP: UDP:
61.42.110.94		Windows 2000 (9)	-	TCP: UDP:
61.76.188.93		64752:108:1414:0:2:1:1:52	-	TCP: UDP:



Network Flows

Timestamp	Direction	Protocol	Src Address	Src Port	Dst Address	Dst Port	Pkts To Dst	Data	Pkts From Dst	Data
2003-08-07 20:08:00	In	TCP	67.100.220.142	4427	192.168.4.2	139	7	524	6	144
2003-08-07 20:08:03	In	TCP	67.100.220.142	4436	192.168.4.2	139	7	524	6	144
2003-08-07 20:08:07	In	TCP	67.100.220.142	4445	192.168.4.2	139	7	524	6	144
2003-08-07 20:08:11	In	TCP	67.100.220.142	4453	192.168.4.2	139	7	524	6	144
2003-08-07 20:08:15	In	TCP	67.100.220.142	4462	192.168.4.2	139	14	1280	12	1225
2003-08-07 20:11:22	In	TCP	33.76.249.77	52435	192.168.4.2	48603	1	0	1	0
2003-08-07 20:12:12	In	TCP	24.247.222.243	4303	192.168.4.2	445	3	0	3	0
2003-08-07 20:13:16	Out	TCP	192.168.4.2	139	217.83.5.40	3380	1	**4**	1	0
2003-08-07 20:13:17	Out	TCP	192.168.4.2	139	67.100.220.142	4462	1	**4**	1	0
2003-08-07 20:17:23	In	TCP	151.20[illegible]69.143	1869	192.168.4.2	445	3	0	3	0
2003-08-07 20:18:18	In	TCP	67.100.220.142	4462	192.168.4.2	139	2	0	1	0
2003-08-07 20:18:20	In	TCP	33.76.249.77	52435	192.168.4.2	48603	1	0	1	0
2003-08-07 20:27:01	In	TCP	62.23.167.250	1761	192.168.4.2	80	3	0	3	0
2003-08-07 20:59:46	In	TCP	65.146.36.67	3134	192.168.4.2	445	3	0	3	0
2003-08-07 21:07:29	In	TCP	65.32.136.23	4262	192.168.4.2	17300	2	0	2	0
2003-08-07 21:10:26	In	TCP	61.241.152.93	4234	192.168.4.2	139	4	0	2	0
2003-08-07 21:10:27	In	TCP	61.241.152.93	4255	192.168.4.2	139	37	4[illegible]7	35	4569
2003-08-07 21:10:27	In	TCP	61.241.152.93	1025	192.168.4.2	139	5	[illegible]2	2	4
2003-08-07 21:10:42	In	TCP	61.241.152.93	1026	192.168.4.2	139	4	[illegible]2	2	4
2003-08-07 21:10:42	In	TCP	61.241.152.93	4557	192.168.4.2	139	7	5[illegible]2	6	144
2003-08-07 21:10:47	In	TCP	61.241.152.93	1027	192.168.4.2	139	5	[illegible]2	2	4
2003-08-07 21:10:47	In	TCP	61.241.152.93	4632	192.168.4.2	139	7	5[illegible]2	6	144

symantec.



Account: aleph1 | **Logout**

Honey Pots : Instances : *738*

[Download Traffic] [Post Comment]

Started (GMT)	2003-08-06 21:12:15
Status	Analyzed
System	35: NT4 workstation #1
Private IP Address	192.168.4.2
Mark Instance	[New] [Known] [**Undefined**]
Reviewer	jroculan

Filesystem Changes	Filesystem Timeline	Remote Systems	Network Flows	Network Events	Instance Events	1 Comment

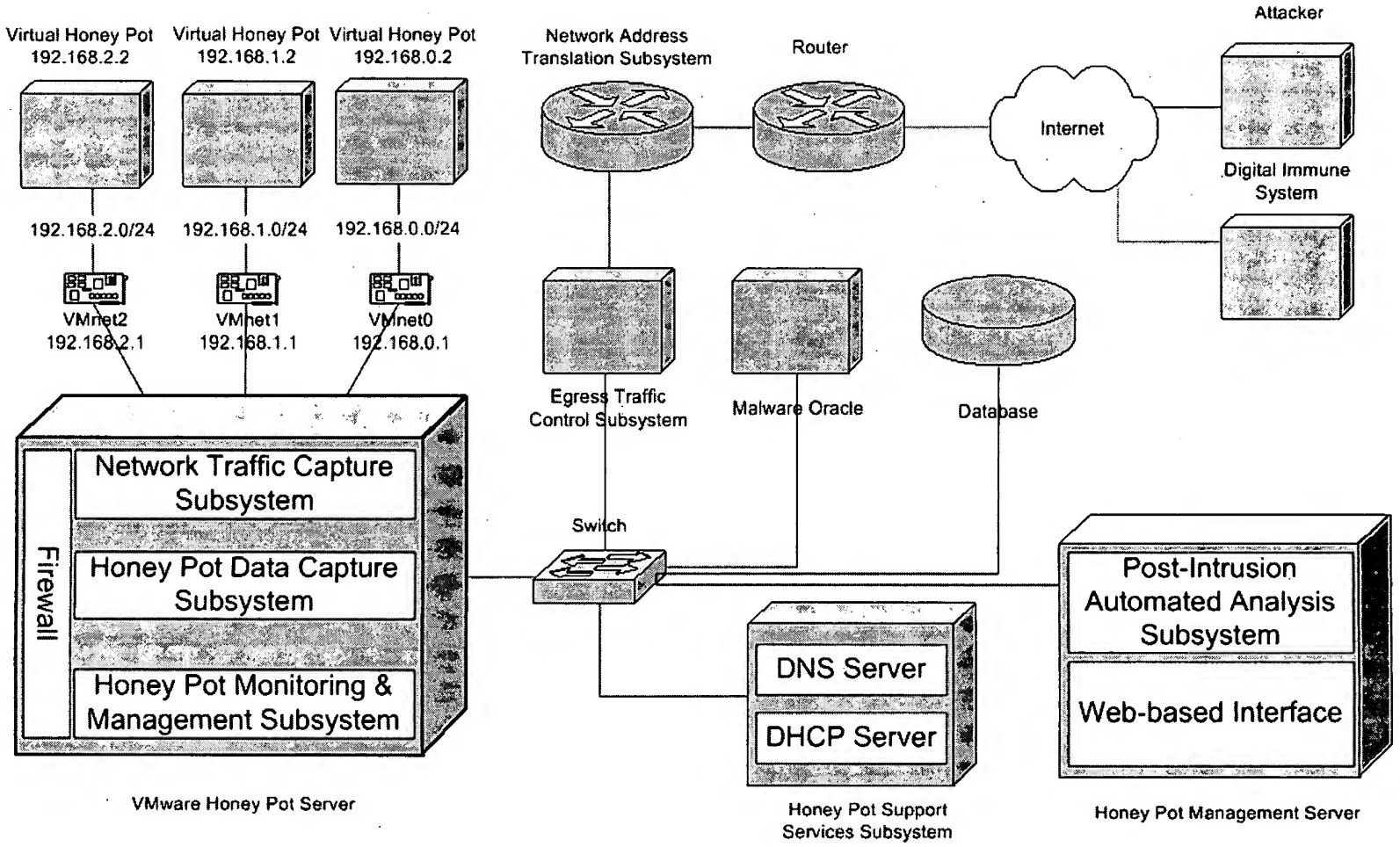
Network Events

Timestamp	Event	Protocol	Src Address	Dst Address	Src Port	Dst Port
2003-08-06 22:31:20	(spp_portscan2) Portscan detected from 192.168.4.2: 4 targets 21 ports in 31 seconds	TCP	192.168.4.2	61.149.130.55	139	4574
2003-08-07 20:06:29	NETBIOS NT NULL session	TCP	67.100.220.142	192.168.4.2	3970	139

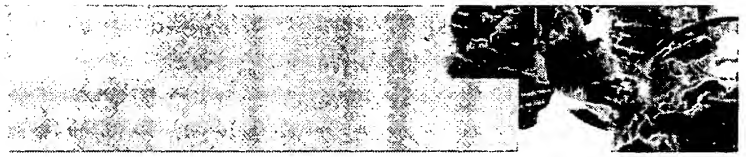
[Start] [Prev] [Next] Page 1 of 1

symantec.

AQS v1.0 Architecture



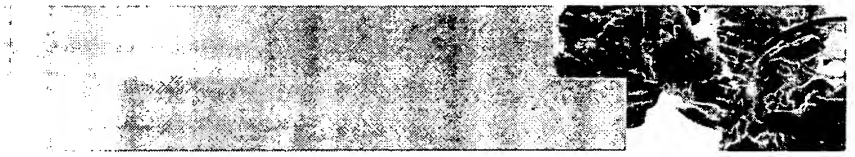
symantec.



AQS v1 Statistics

- Entered production in July 2003.
- Network footprint: 219 IP addresses.
- 1 honey pot server executing 5 to 8 honey pot systems.
- Over 1,500 honey pot instances.
- 63 new malware detected.
  - 17 in July
  - 11 in August
  - 23 in September
  - 12 in October

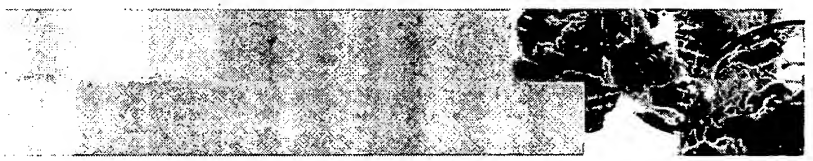
symantec.



AQS Common Offenders

- Welchia
- SQL Slammer
- Lots of variants of Gaobot
- Mumu.A and Mumu.B
- Tzet
- Lots of worms infected by viruses.
- Lots of IRC botnets.

symantec.



AQS v2

- Addition of another honey pot server.
- Network Traffic Control
  - Filter traffic to limit reinfection from the same source.
- Honey Pot Data Capture Subsystem
  - Instrument the honey pot archetypes and capture data from within the honey pot instances.
    - ❖ E.g. keystrokes, all file system access.
- Malware Oracle
  - Detection of non-viral files (e.g. exploits, hacker tools, known good files).
- Inquisitor
  - Actively detect the operating system of attacking hosts.
  - Detect the available services of attacking hosts.

symantec.

Thank you

Questions?